COVID Dashboard with React

In this guided project, you will use a front-end Javascript library called ReactJS to create a dynamic single-page application that visualizes various metrics of COVID data fetched from an API (COVID-19 Canada Open Data Working Group) as a dashboard. Through this guided project you will learn about intergrating various front end libraries and APIs to develop powerful applications with ease.

Learning Objectives

By the end of this guided project, you will be able to :

- Develop a React application from scratch
- Apply and interpret JSX syntax
- Describe and create a React Component
- Utilise React hooks to store data and work with renders/re-renders
- Use third party libraries to streamline development

Prerequisites (optional)

Foundational Javascript, HTML and CSS

Introduction to Working Environment

In this guided project, You will be working in a cloud Integrated Development Environment (IDE) provided by IBM. The IDE does not require installation and comes with a pre-installed package manager allowing you to start the lab without any overhead.

The user interface will be built using a front-end, open source Javascript library called ReactJS. ReactJS is a popular and well maintained library that allows developers to create complex front end applications which can dynamically update without having to reload the whole webpage. Along with optimisations, The library also provides tools and structures that streamline development.

To create a new React application, execute the command below by clicking on the >_ icon.

```
1. 1
```

```
1. npx create-react-app cov-dashboard
```

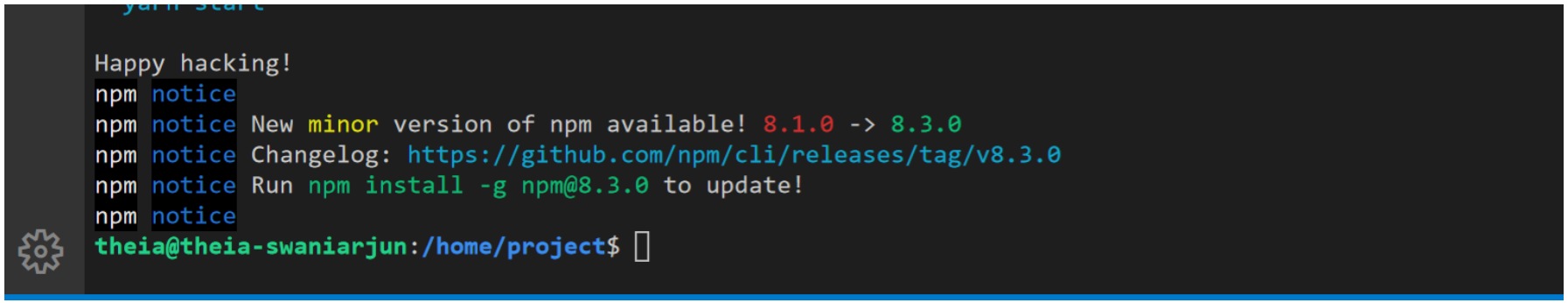
Copied! Executed!

If you see a prompt that says,

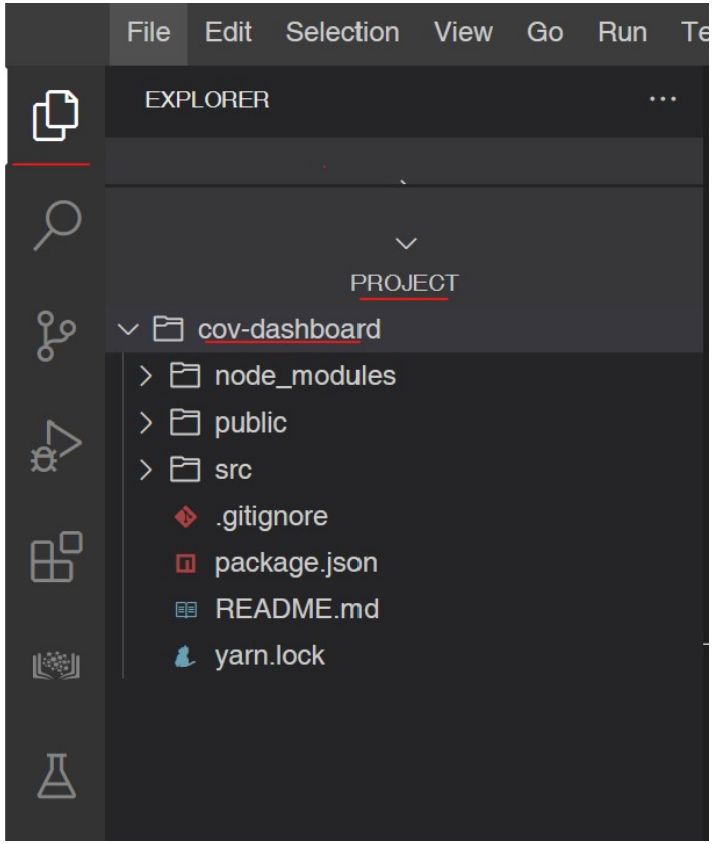
> Need to install the following packages:
> create-react-app

press Enter.

The above command runs the create-react-app tool. You should see a new terminal window the following output.



Create-react-app is a tool that setups the boilerplate code for your application by installing dependencies (other packages ReactJS needs to run) automatically. *cov-dashboard* is the name of your React application. Once the command has run, Click on the top icon in the sidebar to expand it. Expand the project directory by clicking on the second tab in the dropdown. You will see a folder called *cov-dashboard* has been created.



In this guided project, all the files of interest are in the *src* folder.

Navigate into the newly created application folder by executing the command below.

```
1. 1
1. cd cov-dashboard
```

Copied! Executed!

Install the remaining additional libraries by executing the command below.

```
1. 1
1. yarn add chart.js react-chartjs-2 react-select
```

Copied! Executed!

Here, npm is a package manager that allows you to easily install packages published by other developer. The packages installed above are:

- chart.js : chart.js is a library that allows data to be visualised as common graphs by drawing them on HTML Canvas elements.
- react-chartjs-2 : react-chartjs-2 allows the core chart.js library to be used with ReactJS.
- react-select : react-select provides a flexible and customisble dropdown for ReactJS.

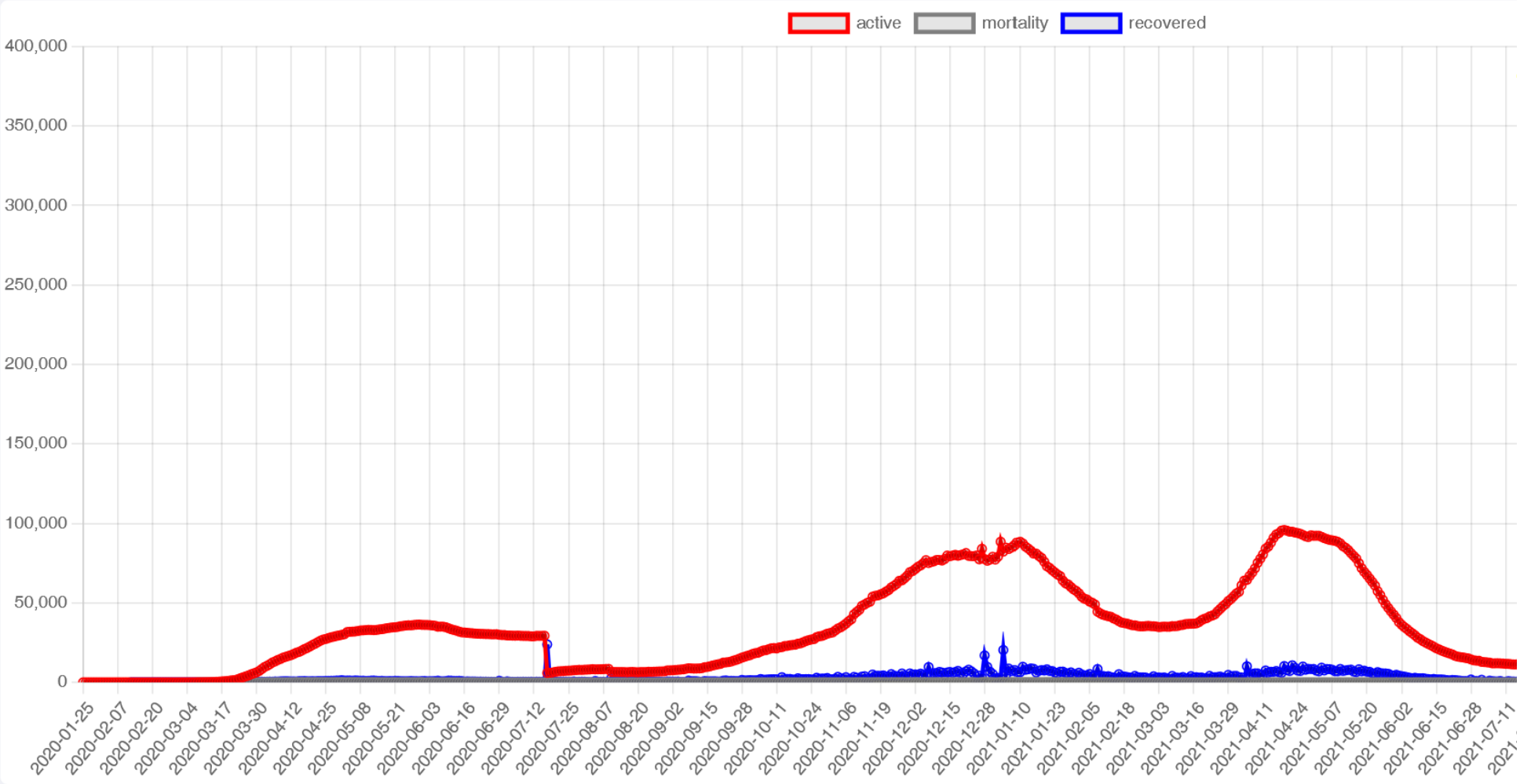
With the setup now complete, You are ready to move on to the next step.

The COVID Dashboard

COVID 19 Dashboard

Canada

1.



Total Cases | Total Recovered | Total Deaths

The screenshot above displays the dashboard you will be developing in this guided project. The dashboard contains three sections:

- The menu bar with a location dropdown and information about the data version
- A line graph displaying timeseries data
- Summary cards showing cumulative data

React JS is an incredibly useful library for developing dynamic user interfaces such as this one. Libraries such as React allow for faster development, under the hood performance optimisations and a high degree of customisation.

The building blocks of a React user interface are independent and re-usable segments called components. Each component is written using JSX - a Javascript extension that allows HTML to be written alongside Javascript. As a result, the visual aspects of the application to be written in the same file as the logic allowing easier flow of information between the two.

Each component can be updated indivually. As a result, rather than re-rendering the whole webpage when the state of a component is updated, React only re-renders the concerned components using its virtual DOM.

In the screenshot above, The dropdown and timeseries graphs are third party components while the summary cards are components you will be making in this lab.

Open *src->App.js* and overwrite its contents with the following :

Open **App.js** in IDE

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
```

```
1. import React from "react";
2. // Imports end
3.
4. function App() {
5. // App component starts here
6. //return statement goes below this
7.   return (
8.     <div className="App">
9.       <h1>COVID 19 Dashboard </h1>
10.       <div className="dashboard-container">
11.         <div className="dashboard-menu"></div>
12.         <div className="dashboard-timeseries"></div>
13.         <div className="dashboard-summary"></div>
14.       </div>
15.     </div>
16.   );
17. }
18.
19. export default App;
```

Copied!

The React library is imported in the first line. The import provides core React functionality such as JSX.

The function `App` is a React Component. It returns JSX which is rendered and displayed on the webpage. Usually, App.js is the parent component for a React App in which other components are nested.

Save the file using *File -> Save*. Finally, run your react application server by running the command below. This starts an application server that lets you view what your app looks like live.

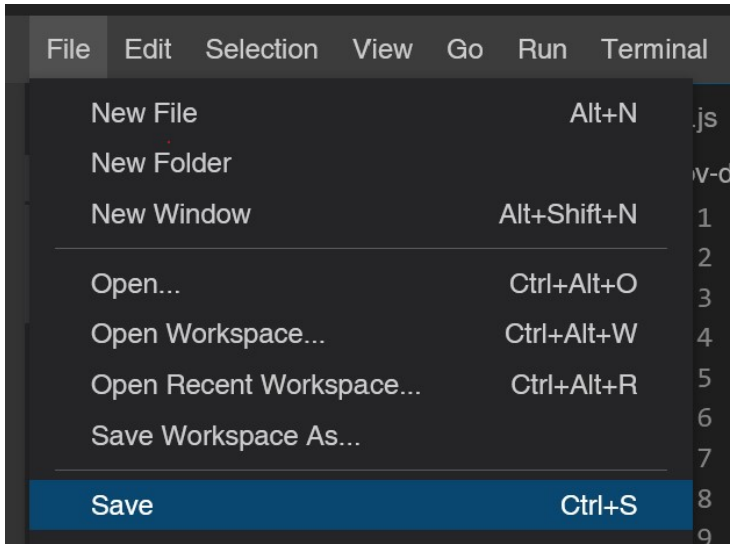
```
1. 1
1. yarn start
```

Copied! Executed!

When you see “webpack compiled successfully” your application is ready to be accessed.

Click the following button to view your application:

Launch Dashboard



COVID 19 Dashboard

top: saving a file. bottom: React Application at end of step

Note:

- There is an alternative class like syntax for React components, however this guided project only utilises the functional component syntax demonstrated above.

Add Style

In this step, a CSS stylesheet will be imported into the application. External stylesheets or script files are usually placed under *src* in the folder structure. During the initialisation using create-react-app, a style sheet called *App.css* was automatically created.

Import *App.css* into the application by opening *App.js* and overwriting the import statements with the code below.

```
1. 1
2. 2
3. 3
```

```
1. import "./App.css";
2. import React from "react";
3. // Imports end
```

Copied!

Open *src -> App.css* and replace the contents of sheet with the code below :

Open **App.css** in IDE

```
 1. 1
 2. 2
 3. 3
 4. 4
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
46. 46
47. 47
48. 48
49. 49
50. 50
51. 51
52. 52
53. 53
54. 54
55. 55
56. 56
57. 57
58. 58
59. 59
```

```
60. 60
61. 61
62. 62
63. 63
64. 64
65. 65
66. 66
67. 67
68. 68
69. 69
70. 70
71. 71
72. 72
73. 73
74. 74
75. 75
76. 76
77. 77
78. 78
79. 79
80. 80
81. 81
82. 82
83. 83
84. 84
```

```
 1. body {
 2.   margin: 0;
 3.   padding: 0;
 4.   color: #242e42;
 5.   background-color: #f0f1f6;
 6.   box-sizing: border-box;
 7.   font-family: sans-serif;
 8.   font-size: 15px;
 9. }
10.
11. h1 {
12.   color: #424656;
13.   text-align: center;
14.   margin-top: 1em;
15.   margin-bottom: 1em;
16. }
17. .dashboard-container {
18.   display: grid;
19.   width: 80vw;
20.   height: 90vh;
21.   grid-template-rows: 0.75fr 5fr 2fr;
22.   grid-gap: 10px;
23.   margin: auto;
24. }
25.
26. .dashboard-container > * {
27.   text-align: center;
28.   border-radius: 5px;
29.   background: white;
30. }
31.
32. .dashboard-menu {
33.   display: flex;
34.   justify-content: space-between;
35.   padding-top: 1em;
36. }
37.
38. .dashboard-loading {
39.   size: 2em;
40. }
41. .dashboard-menu > .dashboard-select {
42.   width: 30%;
43.   margin-left: 1rem;
44. }
45.
46. .dashboard-menu > .update-date {
47.   margin-right: 1rem;
48. }
49. .line-chart {
50.   height: 80%;
51. }
52.
53. .dashboard-summary {
54.   display: flex;
55.   flex-direction: row;
56.   justify-content: space-evenly;
```

```
}

.summary-card {
  flex: 1 0 calc(25% - 10px);
  margin: 5px;
  color: white;
}
.summary-card h2 {
  font-size: 2em;
  padding-top: 5%;
  animation: fade 2s linear;
}
.summary-card p {
  font-size: calc(100% + 1.2vw);
}
.summary-card:nth-child(1) {
  background-color: #8e2e46;
}
.summary-card:nth-child(2) {
  background-color: #1f5673;
}

.summary-card:nth-child(3) {
  background-color: #687987;
}
.summary-card:nth-child(4) {
  background-color: #2a8479;
}
```

Copied!

Save *App.css* and *App.js* and check the dashboard by reloading the tab in the previous step or by clicking the button below.

Launch Dashboard

Your application should now be looking like the image below.

COVID 19 Dashboard

Dashboard Menu

In this step, you will implement the menu section of the dashboard.
The menu bar consists of a location drop down and a data version paragraph. Over the next few steps, you will be implementing the following functionality:

- The drop down should show a list of valid locations accepted by the API. By default, the drop down should have *Canada* selected.
- On selecting a dropdown option, The new selected option should be stored and displayed on the drop down.
- Everytime the location is changed, The data version paragraph should be updated by making a call to the relevant API end point. Later in the project, the data for the summary cards and time series will also be updated on changing the location.

In *App.js*, Amend the import statements to match the following code.

```
1. 1
2. 2
3. 3
4. 4
```

```
1. import "./App.css";
2. import React from "react";
3. import Select from "react-select";
4. // Imports end
```

Copied!

Here a third party component i.e a Select dropdown is imported from one of the libraries installed earlier.

Next update the dashboard-menu div in the App component (inside the App function in *App.js*) like so :

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
1.         <div className="dashboard-menu ">
2.           <Select
3.             options={locationList}
4.             className="dashboard-select"
5.           />
6.           <p className="update-date"> Last Updated :  </p>
7.         </div>
```

Copied!

In the code above, two more elements have been added to the dashboard-menu div. Note how instead of using class, the className is used as an attribute. This is because class is a reserved keyword in Javascript, so JSX tackles this issue by using className instead of the class attribute in HTML.

The Select tag might look like an HTML element, however it is another React component. It is that easy to import and use components in React!

The attributes in React component tags are called *Props* (short for properties). These are used to pass information to components. In this case, *react-select* docs specify what props the Select component accepts and what kind of data can be passed to those props.

The options props accepts an array of objects containing a label and a value key called `locationList`. Note how the Javascript variable is surrounded by curly braces. The curly braces let snippets of Javascript to be embedded into markup.

Finally, define `locationList` in the App component before the return statment like so:

```
1. 1
2. 2
3. 3
4. 4
```

```
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
```

```
1. // App component starts here
2.   const locationList = [
3.     { value: "AB", label: "Alberta" },
4.     { value: "BC", label: "British Columbia" },
5.     { value: "can", label: "Canada" },
6.     { value: "MB", label: "Manitoba" },
7.     { value: "NB", label: "New Brunswick" },
8.     { value: "NL", label: "Newfoundland and Labrador" },
9.     { value: "NT", label: "Northwest Territories" },
10.    { value: "NS", label: "Nova Scotia" },
11.    { value: "NU", label: "Nunavut" },
12.    { value: "ON", label: "Ontario" },
13.    { value: "PE", label: "Prince Edward Island" },
14.    { value: "QC", label: "Quebec" },
15.    { value: "SK", label: "Saskatchewan" },
16.    { value: "YT", label: "Yukon" },
17.  ];
18. //return statement goes below this
```

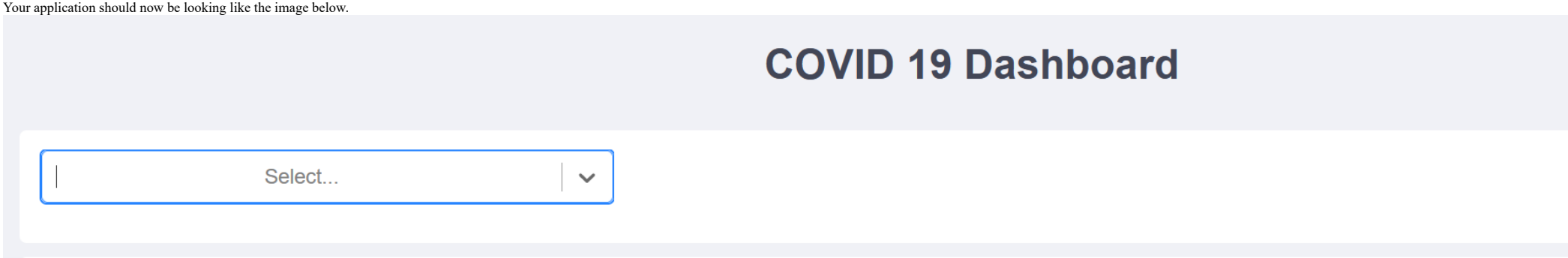
Copied!

The labels correspond to the text displayed in the dropdown and the value corresponds to the value selected on choosing an option. The values were taken from API doc.

Save *App.js* and check the dashboard by reloading the tab in the previous step or by clicking the button below.

Launch Dashboard

Your application should now be looking like the image below.



Storing Active Location with States

A React component re-renders several times in a dynamic application. The values of regular variables are not stored between renders. To keep data between each render, React uses states.

Add the `useState` hook by modifying import statements to the following :

```
1. 1
2. 2
3. 3
4. 4
```

```
1. import "./App.css";
2. import React, { useState } from "react";
3. import Select from "react-select";
4. // Imports end
```

Copied!

Further below the `locationList` declaration add the following:

```
1
2
```

```
const [activeLocation, setActiveLocation] = useState("can");
const [lastUpdated, setlastUpdated] = useState("");
```

Copied!

Hooks in react allow the user to work directly with a components lifecycle (i.e the cycle of it updating and re-rendering). The useState hook declares a new state for the component. Not only does a state preserve information between renders, but a change in state triggers a re-render of the component itself.

The useState accepts the initial value of a state as an argument and returns an array with two items. The first item is the current state value and the second item is a function that updates the state. The array destructuring syntax (as shown above) can be used to store both of these values in easy to read variable names. States corresponding to data version (when the data was last updated) and active location have been declared above.

Finally, update the dashboard-menu div in the App component in *App.js* as follows:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
 <div className="dashboard-menu ">
          <Select
            options={locationList}
            onChange={(selectedOption) =>
              setActiveLocation(selectedOption.value)
            }
            defaultValue={locationList.filter(
              (options) => options.value == activeLocation
            )}
            className="dashboard-select"
          />
          <p classNa me="update-date">
            Last Updated : {lastUpdated}
          </p>
        </div>
```

Copied!

Here, two more props have been added to the Select component :

- `onChange` - Similar to the attribute found in HTML, `onChange` is a function that is triggered when a new option is selected in the dropdown. The function above receives an option object and updates the activeLocation state with value key.
- `defaultValue` - This refers to the initial value of the component. Here all the options are filtered by the value key and the option that matches the initial value of `activeLocation` (in this case “Canada”) remains.

The state should never be modified directly and always through the update function returned by the `useState` hook (for example `setActiveLocation`). This ensures that the component re-renders and updates on state updates.

Save *App.js* and check the dashboard by reloading the tab in the previous step or by clicking the button below.

Launch Dashboard

The page should now render with Canada as the default option for the dropdown.

Fetching Last Updated with useEffect

The last updated information for the endpoint is provided by the maintainers of the API at the `/version` endpoint. On navigating to the endpoint by clicking the link here, you should see a response object the following format -

{
version: xxxx-xx-xx xx:xx EST
}

Create a function that makes a call to this endpoint and updates the `lastUpdated` state using the response. Do so by pasting the code below alongside the `locationList` declaration (inside the app component but above the return statement):

```
1
2
3
4
5
6
```

```
const baseUrl = "https://api.opencovid.ca";
const getVersion = async () => {
  const res = await fetch(`${baseUrl}/version`);
  const data = await res.json();
  setlastUpdated(data.summary);
};
```

Copied!

`getVersion` should be called every time the `activeLocation` state is updated. However, state updates can be asynchronous. So simply putting `getVersion()` call below `setActiveLocation()` would produce erroneous results.

React provides another hook called `useEffect` that allows functions to be called after component re-renders. Since state updates and prop updates are a common source of component re-renders, `useEffect` also provides a mechanism to call functions only after component re-renders due to specific state/prop changes.

Import the `useEffect` hook by modifying import statements to the following :

```
1
2
3
4
```

```
import "./App.css";
import React, { useState, useEffect } from "react";
import Select from "react-select";
// Imports end
```

Copied!

Finally, add the following below the state declarations in the App component in `App.js`

```
1
2
3
```

```
useEffect(() => {
  getVersion();
}, [activeLocation]);
```

Copied!

UseEffect receives two arguments -

- The first argument is a function that is called after a component renders or updates. In this case the function simply calls `getVersion`.
- The second argument is called the *dependency array*. Anytime a state/prop in this array updates (this includes the initial declaration/render), the function in the first argument is run. In this case, the dependant state is activeLocation.

Save *App.js* and check the dashboard by reloading the tab in the previous step or by clicking the button below.

Launch Dashboard

The menu bar should now display an last updated date/time.

Note: The dependency array is an optional argument for useEffect. If it is omitted, The effect is run everytime the component renders/updates. If the array argument is added but the array is left empty, the effect is only run once after the initial render.

Summary Data Component

With the dashboard menu wrapped up, lets move on to the summary card section of the dashboards.

There are four cards in total displaying cumulative cases, recoveries, deaths and vaccinations corresponding to the location selected. The data in these cards should be updated by making a new call to the API every time a new location is selected.
Since the cards are identical in terms of their markup and only differ by the data they are displaying, they are an ideal candidate for a component.

Click the following button to create and open the *SummaryCard.js* file.

Open **SummaryCard.js** in IDE

Then add the following code to the file and save it:

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
```

```
1. import React from "react";
2.
3. export default function Card(props) {
4.   return (
5.     <div className="summary-card">
6.       <h2>{props.title}</h2>
7.       <p>{props.value}</p>
8.     </div>
9.   );
10. }
```

Copied!

Like the previous component, the Card component is a function that returns markup code to be rendered. However, unlike the previous component, Card accepts an argument aptly named props. The props object contains all the props passed down to the component as key value pairs.
In the markup above, The title prop is rendered as a heading and the value prop is rendered as a paragraph.

Save *SummaryCard.js*. The component is now ready to be imported and used.

Add Summary Cards to Dashboard

Open *App.js* and import the *Card* component created in the last step by modifying the imports like so :

```
1. 1
2. 2
3. 3
4. 4
5. 5
```

```
1. import "./App.css";
2. import React, { useState, useEffect } from "react";
3. import Select from "react-select";
4. import Card from "./SummaryCard";
```

```
5. // Imports end
```

Copied!

Declare a new state corresponding the summary card data by adding the following below existing state declarations :

```
1. 1
```

```
1.   const [summaryData, setSummaryData] = useState({});
```

Copied!

Finally, update the dashboard-summary div like so :

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
12. 12
```

```
1.  <div className="dashboard-summary">
2.          <Card title="Total Cases" value={summaryData.cases} />
3.          <Card
4.            title="Total Recovered"
5.            value="not provided"
6.          />
7.          <Card title="Total Deaths" value={summaryData.deaths} />
8.          <Card
9.            title="Total Vaccinated"
10.           value={summaryData.vaccine_administration_total_doses}
11.         />
12.       </div>
```

Copied!

Here, The card component is called with title and value props. Since no keys for `summaryData` exist (it was declared with an initial state of {}), the value props are undefined and will render as empty strings in the component paragraphs.

Save *App.js* and check the dashboard by reloading the existing application tab or by clicking the button below.

Launch Dashboard

You should see the following ouput :

COVID 19 Dashboard

Select...

L

Total Cases

Total Recovered

Total Deaths

To

Fetching Summary Data

To get the data for the summary cards, a call to the `/summary` endpoint is made. Open the link here to see the response. You should see a JSON object like the following :


```
▼ summary:
  ▼ 0:
      active_cases:           43414
      active_cases_change:    0
      avaccine:               0
      cases:                  0
      cumulative_avaccine:    7825022
      cumulative_cases:       395252
      cumulative_cvaccine:    3223444
      cumulative_deaths:      3338
      cumulative_dvaccine:    9480789
      cumulative_recovered:   348500
      cumulative_testing:     6480159
      cvaccine:               0
      date:                   "08-01-2022"
      deaths:                 0
      dvaccine:               0
      province:               "Alberta"
      recovered:              0
      testing:                0
      testing_info:           "NULL"
  ▶ 1:                        {…}
  ▶ 2:                        {…}
  ▶ 3:                        {…}
  ▶ 4:                        {…}
```


The endpoint above returns summary information for all recorded locations (each numerical key corrresponds to a location). To get the summary location for a paticular location, a loc parameter can be added to the url along with the endpoint (e.g. /summary?loc=Canada).

Add another function to fetch SummaryData below `getVersion()` by adding the code below to the App component.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
```

```
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
```

```
1.   const getSummaryData = async () => {
2.     let res;
3.     if (activeLocation === "can") {
4.         res = await fetch(`${baseUrl}/summary?geo=${activeLocation}`);
5.     } else {
6.         res = await fetch(`${baseUrl}/summary?loc=${activeLocation}`);
7.     }
8.     let resData = await res.json();
9.     console.log(resData)
10.    let summaryData = resData.data[0];
11.
12.    console.log(summaryData)
13.    let formattedData = {};
14.
15.    Object.keys(summaryData).map(
16.      (key) => (formattedData[key] = summaryData[key].toLocaleString())
17.    );
18.    console.log(formattedData)
19.    setSummaryData(formattedData);
20.  };
```

Copied!

In the function above, The data is fetched by making a call to the `/summary` endpoint with the loc parameter set to `activeLocation`, The string is then formatted using `toLocaleString()` (which in this case adds commas to the hundreds and thousands position). Finally, the `summaryData` state is updated with the formatted data.

Update the useEffect hook to add `getSummaryData()` as follows -

```
1. 1
2. 2
3. 3
4. 4
```

```
1.   useEffect(() => {
2.     getSummaryData();
3.     getVersion();
4.   }, [activeLocation]);
```

Copied!

Launch Dashboard

You should now see data in the summary cards. Change locations and ensure data updates.

Optional:

You might notice the lag in the new data rendering caused by the time taken to fetch the data. It is usually good practice to visually represent expected delays to let the user know the application is working. This can simply be done by clearing the old data while the new data is being rendered. Try to implement this on your own.

▶ Click here for the answer

Timeseries Data

This leaves the last section of the dashboard - the time series graph.

Open *App.js* and modify the imports like so -

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
1. import "./App.css";
2. import React, { useState, useEffect } from "react";
```

```
import Select from "react-select";
import Card from "./SummaryCard";
import { Line } from "react-chartjs-2";
import Chart from "chart.js/auto";
// Imports end
```

Copied!

The first new line imports the Line graph component. The second new line imports additional components/utilities in Graph.js used by Line.

To complete the implementation of this section, you will -

- Declare a state that holds the data for the Line graph
- Create a new Line graph component and pass data and graph options as props
- Fetch data from the /timeseries API endpoint and map it a format required by the Line component
- Update the state holding data for the Line graph after API call

Create a new state declaration to store timeseries data by pasting the following below existing state declarations :

```
1
2
3
```

```
const [timeseriesData, setTimeseriesData] = useState({
  datasets: [],
});
```

Copied!

The initial state has a datasets key as it is required by the Line component.

Paste the following below `baseUrl` constant declaration :

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

```
const timeseriesOptions = {
  responsive: true,
  normalized: true,
  plugins: {
    tooltip: {
      enabled: false,
    },
  },
  maintainAspectRatio: false,
  scales: {
    y: {
      min: 0,
    },
  },
};
```

Copied!

The `tsOptions` object will be passed as a prop to the Line graph component. The component API specifies the options that can be customized for the chart along with accepted parameters.

Finally add the Line graph component by updating dashboard-timeseries div :

```
1
2
3
4
5
6
7
```

```
1.        <div className="dashboard-timeseries">
2.          <Line
3.            data={timeseriesData}
4.            options={timeseriesOptions}
5.            className="line-chart"
6.          />
7.        </div>
```

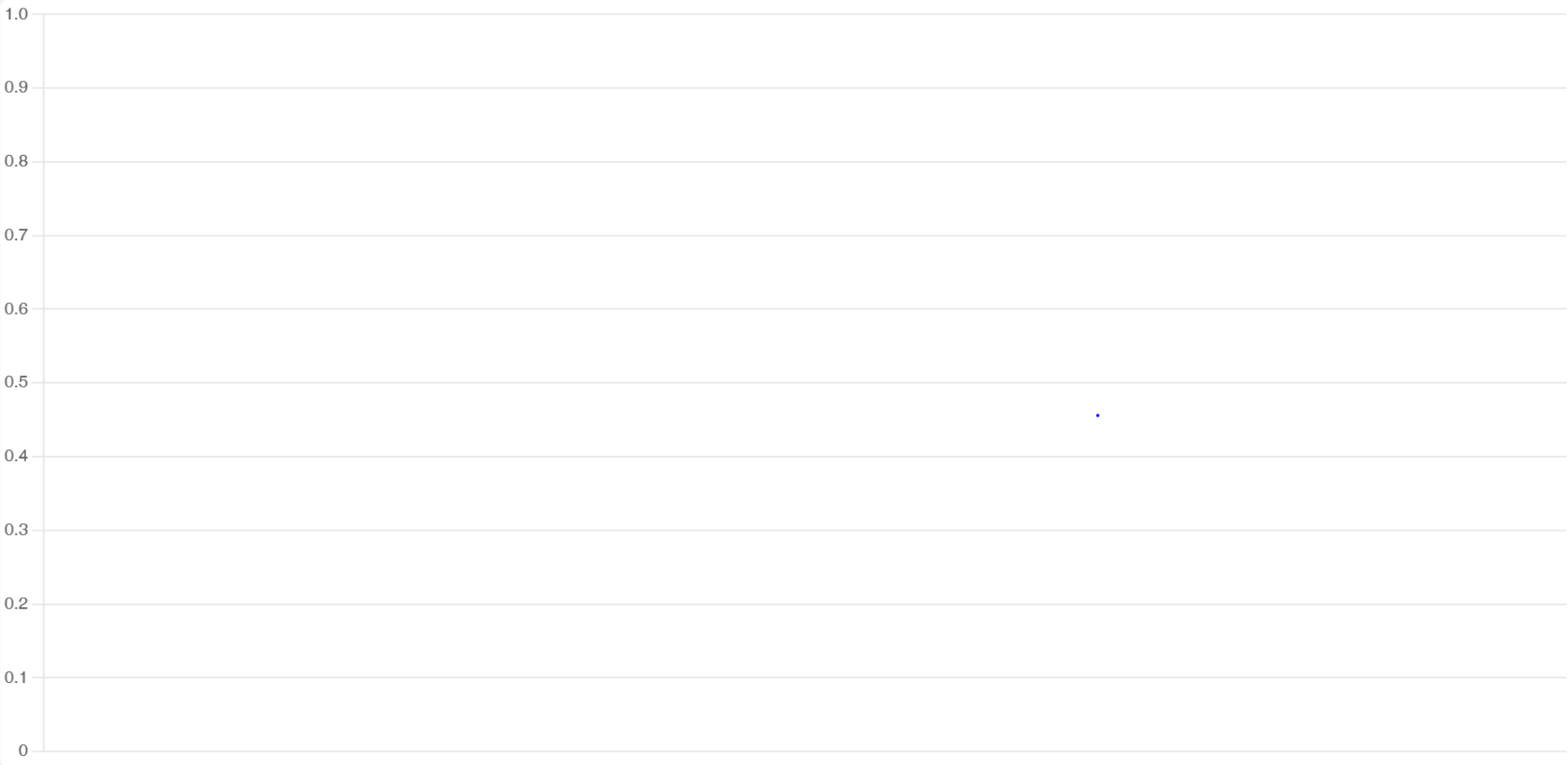
Copied!

Launch Dashboard

You should now see the following output.

COVID 19 Dashboard

Canada



Map Time Series Data

The data for the time series will be retrieved from the `\timeseries` endpoint. Click on the link here and see the response retrieved.

```
▶ active:      […]
▶ avaccine:    […]
▶ cases:       […]
▶ cvaccine:    […]
▶ dvaccine:    […]
▶ mortality:   […]
▶ recovered:   […]
▶ testing:     […]
```

```
▼ active:
  ▼ 0:
      active_cases:           0
      active_cases_change:    0
      cumulative_cases:       0
      cumulative_deaths:      0
      cumulative_recovered:   0
      date_active:            "25-01-2020"
      province:               "Alberta"
  ▶ 1:                        {…}
  ▶ 2:                        {…}
  ▶ 3:                        {…}
  ▶ 4:                        {…}
  ▶ 5:                        {…}
  ▶ 6:                        {…}
```

top: response (collapsed)
bottom: response (partially expanded)

The retrieved object has keys corresponding to different time series where each key stores an array of data points. The data points are objects storing information about the date, the value of the series on the day, location and other cumulative/differential data.

The data can be filtered by location using the `loc` parameter like in summary data. In additon, another parameter called `ymd` will be set to `true` while making the API to ensure that date references in the returned data follow the yyyy-mm-dd format preferred by the line component.

For the data to be usable by line component it has to be mapped into a different format. Each data series or dataset should have the following structure:

```
1. 1
2. 2
3. 3
```

```
4. 4
5. 5
6. 6
7. 7
8. 8
9. 9
10. 10
11. 11
```

```
1. {
2. label  :  Name of the dataset,
3. borderColor : Color on graph,
4. data : [ Array containing information from each data point in the series
5.     {
6.         x : Date of the datapoint
7.         y : Data to be plotted
8.     },
9.     ...
10.         ]
11. }
```

Copied!

In this guided project, The three data sets of interests are as follows:

- *Label* : active
  Border color : red
  Datapoint Y : active_cases
  Datapoint X : date_active

- *Label* : mortality
  Border color : grey
  Datapoint Y : cumulative_deaths
  Datapoint X : date_death_report

- *Label* : recovered
  Border color : blue
  Datapoint Y : recovered
  Datapoint X : date_recovered

To map the data, A function called `timeseriesDataMap` has to be implemented that :

- Accepts the response for `/timeseries` endpoint as an argument
- Maps each of the relevant series (data sets of interest) into the data series format specified above
- Returns an array of all of the mapped datasets

As an optional excercise, Implement the `timeseriesDataMap` function on your own:

▶ Click here for the answer

Note : The Graph.js library is extensive and fairly customisable. There may be multiple ways the API data retrieved in this step can be mapped into a form usable by the Line component.

Fetch Time Series Data

In this step, you will be wrapping up the Time Series section of the dashboard. Try to implement the remaining functionality on your own using the hints below. Feel free to view the solution if you get stuck.

First, make a function that :

- Fetches data from the `/timeseries?loc=ActiveLocation&ymd=true` endpoint
- Maps the data using `timeseriesDataMap` and stores it in an object with the following structure : `{ datasets: mappedData }`
- Updates the `timeseriesData` state with the object in the last step

Hint : When updating a React state, make sure you do not mutate the existing state directly. This is a common pitfall when dealing when React state objects. For example:

Let newTimeseriesData= timeseriesData
newTimeseriesData[datasets] = mappedData
setTimeseriesData(newTimeseriesData)

The above example is incorrect, as it creates a variable `newTimeseriesData` that references the existing state `timeseriesData` and then mutates it directly.

Try to implement the function described above on your own.

▶ Click here for the answer

Finally, ensure that `getTimeseriesData` is called everytime there is an update in the `activeLocation` state.

Try to implement the functionality described above on your own.

▶ Click here for the answer

Save *App.js* before moving on to the next step.

All Done!

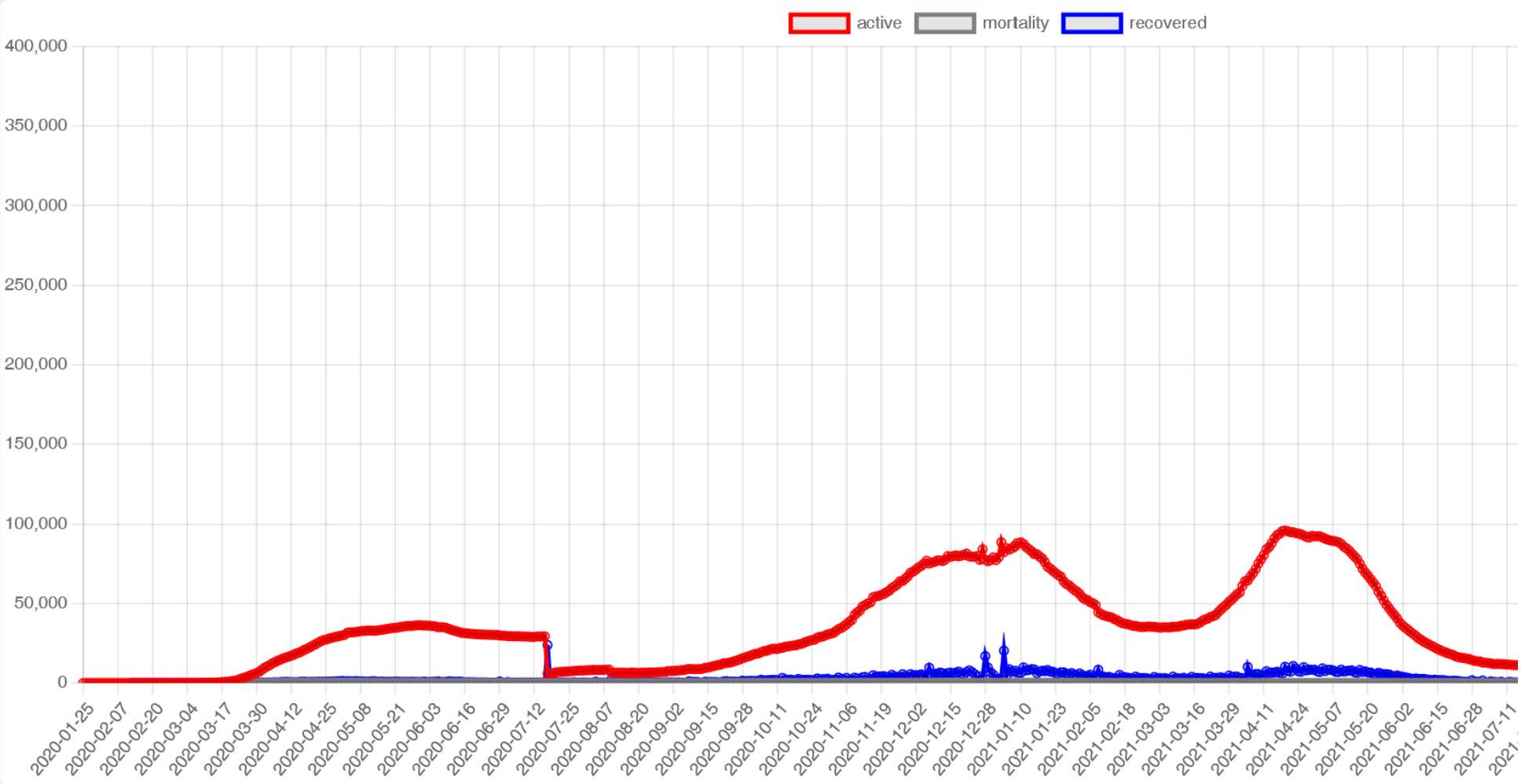
Congratulations! You have reached the end of the guided project. Open up the application one last time and ensure that you see a dashboard like the one below.

Launch Dashboard

COVID 19 Dashboard

Canada

1.



Total Cases | Total Recovered | Total Deaths

In case you get stuck, reference your code against the code here.

```
 1. 1
 2. 2
 3. 3
 4. 4
 5. 5
 6. 6
 7. 7
 8. 8
 9. 9
10. 10
11. 11
12. 12
13. 13
14. 14
15. 15
16. 16
17. 17
18. 18
19. 19
20. 20
21. 21
22. 22
23. 23
24. 24
25. 25
26. 26
27. 27
28. 28
29. 29
30. 30
31. 31
32. 32
33. 33
34. 34
35. 35
36. 36
37. 37
38. 38
39. 39
40. 40
41. 41
42. 42
43. 43
44. 44
45. 45
46. 46
47. 47
48. 48
49. 49
50. 50
51. 51
52. 52
53. 53
54. 54
55. 55
56. 56
57. 57
58. 58
59. 59
60. 60
61. 61
62. 62
```

63. 63
64. 64
65. 65
66. 66
67. 67
68. 68
69. 69
70. 70
71. 71
72. 72
73. 73
74. 74
75. 75
76. 76
77. 77
78. 78
79. 79
80. 80
81. 81
82. 82
83. 83
84. 84
85. 85
86. 86
87. 87
88. 88
89. 89
90. 90
91. 91
92. 92
93. 93
94. 94
95. 95
96. 96
97. 97
98. 98
99. 99
100. 100
101. 101
102. 102
103. 103
104. 104
105. 105
106. 106
107. 107
108. 108
109. 109
110. 110
111. 111
112. 112
113. 113
114. 114
115. 115
116. 116
117. 117
118. 118
119. 119
120. 120
121. 121
122. 122
123. 123
124. 124
125. 125
126. 126
127. 127
128. 128
129. 129
130. 130
131. 131
132. 132
133. 133
134. 134
135. 135
136. 136
137. 137
138. 138
139. 139
140. 140
141. 141
142. 142
143. 143
144. 144

```
145. 145
146. 146
147. 147
148. 148
149. 149
150. 150
151. 151
152. 152
153. 153
154. 154
155. 155
156. 156
157. 157
158. 158
159. 159
160. 160
161. 161
162. 162
163. 163
164. 164
165. 165
166. 166
```

```
 1. import "./App.css";
 2. import Select from "react-select";
 3. import React, { useState, useEffect } from "react";
 4. import Card from "./SummaryCard";
 5. import { Line } from "react-chartjs-2";
 6. import Chart from "chart.js/auto";
 7.
 8. function App() {
 9.   const locationList = [
10.     { value: "AB", label: "Alberta" },
11.     { value: "BC", label: "British Columbia" },
12.     { value: "canada", label: "Canada" },
13.     { value: "MB", label: "Manitoba" },
14.     { value: "NB", label: "New Brunswick" },
15.     { value: "NL", label: "Newfoundland and Labrador" },
16.     { value: "NT", label: "Northwest Territories" },
17.     { value: "NS", label: "Nova Scotia" },
18.     { value: "NU", label: "Nunavut" },
19.     { value: "ON", label: "Ontario" },
20.     { value: "PE", label: "Prince Edward Island" },
21.     { value: "QC", label: "Quebec" },
22.     { value: "SK", label: "Saskatchewan" },
23.     { value: "YT", label: "Yukon" },
24.   ];
25.   const baseUrl = "https://api.opencovid.ca";
26.   const timeseriesOptions = {
27.     responsive: true,
28.     normalized: true,
29.     plugins: {
30.       tooltip: {
31.         enabled: false,
32.       },
33.     },
34.     maintainAspectRatio: false,
35.     scales: {
36.       y: {
37.         min: 0,
38.       },
39.     },
40.   };
41.
42.   const [activeLocation, setActiveLocation] = useState("canada");
43.   const [lastUpdated, setlastUpdated] = useState("");
44.   const [summaryData, setSummaryData] = useState({});
45.   const [timeseriesData, setTimeseriesData] = useState({
46.     datasets: [],
47.   });
48.
49.   useEffect(() => {
50.     getVersion();
51.     getSummaryData();
52.     getTimeseriesData();
53.   }, [activeLocation]);
54.
55.   const getVersion = async () => {
56.     const res = await fetch(`${baseUrl}/version`);
57.     const data = await res.json();
58.
59.     setlastUpdated(data.version);
```

```
60.   };
61.   const getSummaryData = async (location) => {
62.     setSummaryData({});
63.     let res = await fetch(`${baseUrl}/summary?loc=${activeLocation}`);
64.     let resData = await res.json();
65.     let summaryData = resData.summary[0];
66.     let formattedData = {};
67.
68.     Object.keys(summaryData).map(
69.       (key) => (formattedData[key] = summaryData[key].toLocaleString())
70.     );
71.     setSummaryData(formattedData);
72.   };
73.   const getTimeseriesData = async (location) => {
74.     const res = await fetch(
75.       `${baseUrl}/timeseries?loc=${activeLocation}&ymd=true`
76.     );
77.
78.     const data = await res.json();
79.
80.     setTimeseriesData({ datasets: timeseriesDataMap(data) });
81.   };
82.   function timeseriesDataMap(fetchedData) {
83.     let tsKeyMap = [
84.       {
85.         datasetLabel: "active",
86.         dataKey: "active_cases",
87.         dateKey: "date_active",
88.         borderColor: "red",
89.       },
90.       {
91.         datasetLabel: "mortality",
92.         dataKey: "deaths",
93.         dateKey: "date_death_report",
94.         borderColor: "grey",
95.       },
96.       {
97.         datasetLabel: "recovered",
98.         dataKey: "recovered",
99.         dateKey: "date_recovered",
100.        borderColor: "blue",
101.      },
102.    ];
103.
104.    let datasets = [];
105.    tsKeyMap.forEach((dataSeries) => {
106.      let dataset = {
107.        label: dataSeries.datasetLabel,
108.        borderColor: dataSeries.borderColor,
109.        data: fetchedData[dataSeries.datasetLabel].map((dataPoint) => {
110.          return {
111.            y: dataPoint[dataSeries.dataKey],
112.            x: dataPoint[dataSeries.dateKey],
113.          };
114.        }),
115.      };
116.      datasets.push(dataset);
117.    });
118.
119.    return datasets;
120.  }
121.
122.  return (
123.    <div className="App">
124.      <h1>COVID 19 Dashboard </h1>
125.
126.      <div className="dashboard-container">
127.        <div className="dashboard-menu ">
128.          <Select
129.            options={locationList}
130.            onChange={(selectedOption) =>
131.              setActiveLocation(selectedOption.value)
132.            }
133.            defaultValue={locationList.filter(
134.              (options) => options.value == activeLocation
135.            )}
136.            className="dashboard-select"
137.          />
138.          <p classNa me="update-date">
139.            Last Updated : {lastUpdated}
140.          </p>
141.        </div>
```

```
        <div className="dashboard-timeseries">
          <Line
            data={timeseriesData}
            options={timeseriesOptions}
            className="line-chart"
          />
        </div>
        <div className="dashboard-summary">
          <Card title="Total Cases" value={summaryData.cumulative_cases} />
          <Card
            title="Total Recovered"
            value={summaryData.cumulative_recovered}
          />
          <Card title="Total Deaths" value={summaryData.cumulative_deaths} />
          <Card
            title="Total Vaccinated"
            value={summaryData.cumulative_avaccine}
          />
        </div>
      </div>
    </div>
  );
}

export default App;
```

Copied!

Summary

In this lab you have learned how to :

- Develop a React application from scratch using create-react-app
- Apply and interpret JSX syntax to couple visual and logic elements in an application
- Describe and create a React Component that can accept and manipulate props
- Utilise React hooks to store data through useSate and work with renders/re-renders through useEffect
- Apply third party libraries such as chart.js and create-react-select to streamline development