COVID-19 Dashboard Developer Manual

By: DreamTeam

Table of Contents

Legal Disclaimer

Local Setup

Installing Node

Navigate to https://nodejs.org/en/download/ and select the appropriate installer for your computer. Follow the on-screen instructions.

Cloning Repository

The repository is available at https://github.com/balogunb/DB_WebPage. You can clone the repository using the command:

```
git clone git@github.com:balogunb/DB_WebPage.git
```

The frontend node app can be found in the folder vue_frontend/ and the backend can be found in the folder backend/

Installing Dependencies
Navigate to the frontend folder "vue_frontend/' and run the following command to install all of the dependencies needed for the front end:

```
npm install
```

Navigate to the backend folder "backend/' and run the following command to install all of the dependencies needed for the backend:

```
npm install
```

Starting Node Server

Navigate to vue_frontend/ . Run the following commands to install axios and launch the local node server.

```
npm run serve
```

Once successfully compiled and launched, Node will provide URLs to access the website at. The default local url is http://localhost:8080/. The data on the website is dependent on api calls to the dream team server at 139.147.9.191:80

Note: While the frontend in the folder 'vue_frontend/' can be run locally, the backend cannot be run locally because it depends on an sql database on the dream team server which cannot be accessed outside of the server.

Database Setup

The backend is dependent on a PostgreSQL database. To duplicate the database proceed with the following steps.

Installing PostgreSQL

Navigate to https://www.postgresql.org/download/ and select the appropriate installer for your computer. Follow the on-screen instructions.

Load data into PostgreSQL

The submission contains a custom dump file 'dreamteamdb.dmp' which can be used to duplicate the database used for creating the REST API. Use the following commands to restore the database.

```
psql  dbname < dreamteamdb.dmp
```

With this you now have an exact copy of the database used in creating the REST API.

Front End

Vue.js

Our webpage uses Vue.js, one of the fastest-growing frontend JavaScript frameworks. We chose Vue over competitors like Angular and React because written Vue clearly distinguishes between HTML, JavaScript, and Vue-specific functionality in a way that Angular and React (particularly JSX) do not. As a consequence, learning Vue is more about *how Vue's added functionality works* than about fighting a whole new "language". Vue allows us to make our webpage highly responsive to user input and database changes thanks to features like property binding.

Axios

Axios is a JavaScript library for asynchronously making HTTP requests using promises. Early in the development of the frontend, We chose Axios because we saw that it was a popular choice in tutorials involving Vue apps. The most time-consuming part of the early stages of development was learning how to integrate external libraries with Vue, so having examples to draw from was a key concern.

Chart.js

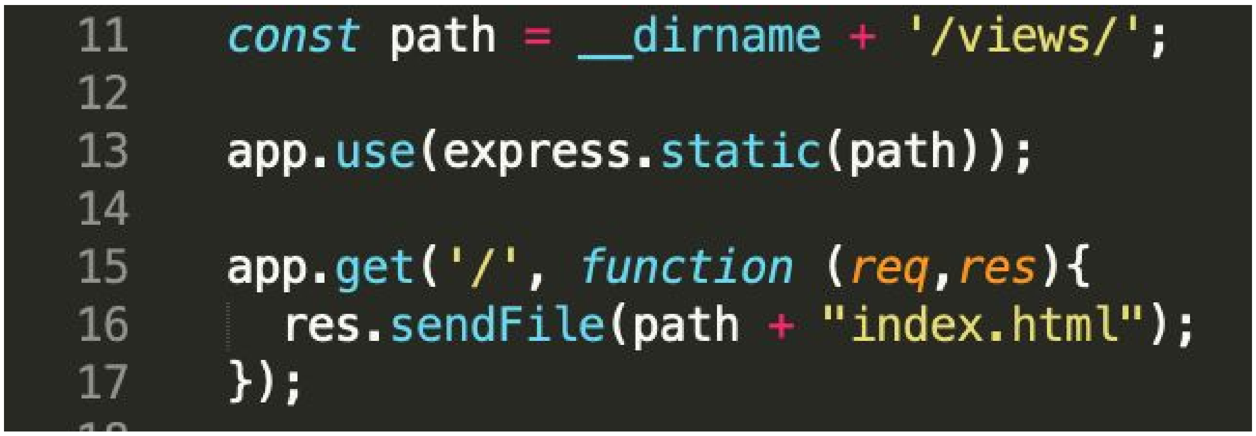
Chart.js is a great open source library that plots data on websites. We chose to use Chart.js because of its clean, object-oriented approach to chart datasets and the existence of *vue-chartjs*, a library that provides Chart.js charts packaged as Vue components.

Back End

REST API (Express.js)
Express.js is a fast web framework for Node.js and Vue. It offers many features that make web development easy. It was easy to configure and customize different routes based on HTTP methods and URLs. It also helped create our REST API server and connect them to our PostgreSQL databases.

For our implementation, we created a REST API which serves all the data we needed from the database. In addition to providing the database, it is also important to note that our REST API also servers our static website. The static file is stored in the location backend/views.

In order to get the static website, we build our vue_frontend using ‘npm run build’ which saves the static build of the vue_website to the build folder in ‘vue_frontend’. These file are then moved to the ‘backend/view’ location. The index.html file in the view folder is then served as shown in the image below.

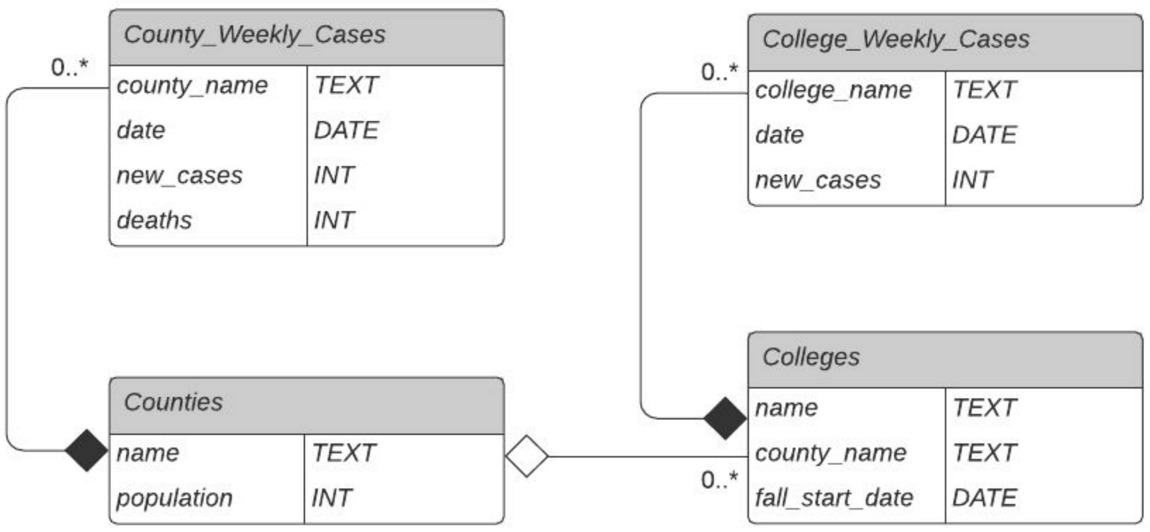


```
const path = __dirname + '/views/';

app.use(express.static(path));

app.get('/', function (req,res){
  res.sendFile(path + "index.html");
});
```

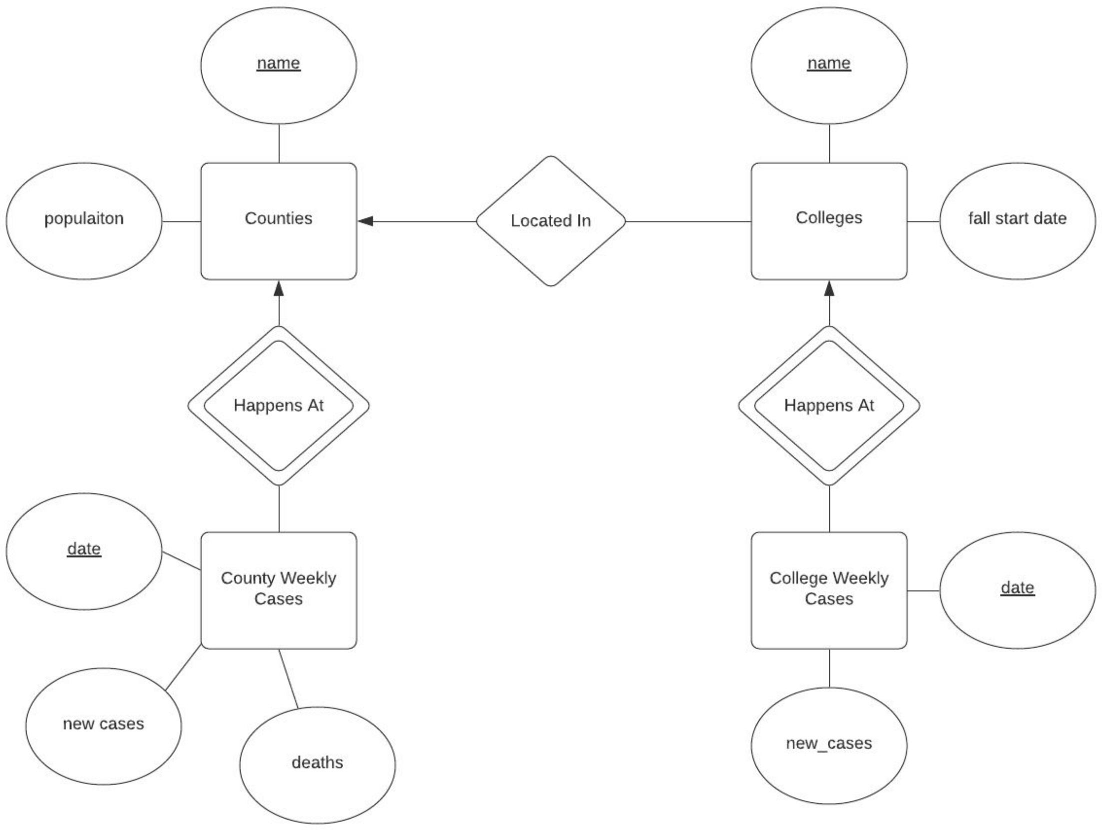
Database
The data is stored in a PostgreSQL database. The database uses the following Schema.

```
CREATE TABLE counties (
      name TEXT PRIMARY KEY,
      population INT
);

CREATE TABLE colleges (
      name TEXT PRIMARY KEY,
      county_name TEXT,
      fall_start_date DATE,
      CONSTRAINT fk_college_counties FOREIGN KEY (county_name)
REFERENCES counties(name)
);

CREATE TABLE county_weekly_cases (
      county_name TEXT,
      date DATE,
      new_cases INT,
      new_deaths INT,
      PRIMARY KEY(county_name, date),
      CONSTRAINT fk_case_counties FOREIGN KEY (county_name)
      REFERENCES counties(name)
);

CREATE TABLE college_weekly_cases (
      college_name TEXT,
      date DATE,
      new_cases INT,
      PRIMARY KEY(college_name, date),
      CONSTRAINT fk_case_colleges FOREIGN KEY (college_name)
      REFERENCES colleges(name)
);
```

Database Schema



Database Design



Data Sources

1. **County Level Data:**
    a. **Links:**
    https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/state/pennsylvania
    https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/state/new-jersey
    b. **How to access**: To access the county level data from the link above, click the link and click on the county you want to get data on from the table. From the new page you can get data in new cases per day and new death per day from the two graphs shown, by hovering over the graph. The data is collected weekly so the data inserted into the database would be the sum of new cases/new death over the week.
2. **College Level Data:**
    a. **Links:**
        i. https://covid19.lafayette.edu/covid-19-dashboard/
        ii. https://coronavirus.lehigh.edu/covid-19-dashboard-reporting
        iii. https://www.moravian.edu/fall-2020/covid-dashboard
        iv. https://www.desales.edu/inside-desales/emergency-updates/covid-19-statistics
        v. https://cedarcrest.edu/healthservices/covid.shtm#community
        vi. https://app.powerbi.com/view?r=eyJrIjoiMDFhMzI2YzQtNmQwNC00YjgzLWFjMzAtZmFlNGQyZGZiZGJhIiwidCI6IjdjZjQ4ZDQ1LTNkZGItNDM4OS1hOWMxLWMxMTU1MjZlYjUyZSIsImMiOjF9
    b. **How to access**: Access of the data differs slightly depending on the college you are looking to get the data from using the link above. Some schools report data daily while some schools report data weekly. If the data is weekly then that can simply be used for the database but if the data is reported daily we first need to add the sum of all the new cases over the week before inputting it in association with the last day of the week into the database. The start date can be gotten from looking at the first day they colleges started reporting covid-19 Cases since all the colleges start reporting cases from the first day.
3. **County Population Data:**
    a. **Link:**https://www.census.gov/data/tables/time-series/demo/popest/2010s-counties-total.html
    b. **How to access**: To access the data, open the link and download the excel file for counties in pennsylvania by clicking pennsylvania. In the table, you can get the county population data by looking for the county name and checking the column(attribute) “census” to get the population count for the desired county.

Advanced Troubleshooting

If the charts are not displaying any information, look at the routes that don't require a request in the appendix and paste the route into a web browser. The requested information in JSON format should appear, indicating that the HTTP get request was successful. If it doesn't appear, check the status of the database.

Extending the System

Limitations of Current System

The user cannot scale x-axis to zoom into specific weeks and days that they would like to see. Also we don't have axis labels on any of our graphs. Furthermore, it should be noted that the "Lehigh Valley" metrics for case numbers and incidence are coded to incorporate data from every county in the database. If new counties are added to the database, their data will be integrated into the "Lehigh Valley" metrics on the frontend webpage. Distinguishing between Valley and non-Valley counties will require an expansion of the database schema.

How to Extend the System

For the most part, extending the system can be done through the database. If new counties and colleges are added to the database, the menu and charts on the webpage will automatically expand to properly incorporate them upon the next page load.

Inserting Another Chart

In App.vue, in the template tag:

```
<RandomChart :customdata="NEW_CHART_DATA" :stack="true/false"
:title="NEW_CHART_TITLE"></RandomChart>
```

Stacking stacks the y values for each point on top of one another; stacking is easier to visualize if the fill is set to true later on.

In App.vue, in the script tag:

Under data, add lines:

```
NEW_CHART_DATA: {},
...
NEW_CHART_TITLE: "New Chart Title",
```

Under watch["selected"], add data information that should change based on the county dropdown. Under mounted, add functions that should run once on startup.

```
let datasetOnChart = {
```

```
label: "Data label",
data: arrayOfData,
backgroundColor: '#hex_code',
borderColor: '#hex_code',
fill: true/false
}
...
this.NEW_CHART_DATA = {
labels: xAxisLabels,
datasets: [dataset1OnChart, dataset2OnChart, ...]
}
```

To add instance-level customizations to the Vue RandomChart component, add new elements the “props” property of RandomChart, and modify the “opts” variable to take the new prop into account, as seen here:

```
      props: ['stack', 'customdata', 'displayAverage',
'displayFloats', 'title'], //passed from parent as v-binds
       data () {
             return {
                   datacollection: this.customdata, //from props
                   opts: {
                         tooltips: {
                               mode: 'index',
                               intersect: false,
                               callbacks: {}
                         },
                         hover: {
                               mode: 'index',
                               intersect: false
                         },
                         scales: {
                               yAxes: [{
                                     stacked: this.stack //from props
                               }]
                         },
                         title: {
                            display: true,
                            text: this.title, //from props
                            fontSize: 20,
                            fontColor: 'black'
                        }
                   }
              }
         }
```
The contents of the *opts* variable are used as the *options* of the chart.js chart. To get a sense of what you can customize, see the official documentation here: https://www.chartjs.org/docs/latest/

Fetching Data
Axios has been aliased to Vue.prototype.$http. To make a new GET request, modify the following code:

```
//request
this.$http.get('http://139.147.9.191:80/[ROUTENAME]', {params:
    { //this dict can have as many parameters as the route requires
        [PARAMETER 1 NAME] : [PARAMETER 1 VALUE]
    }
}).then(response => {
  // the operations to perform on the returned 'response' object
})
```
Because Axios requests are made asynchronously, anything you want to do with the response should be done in the function you pass to the *.then*() call after the request. If you need to use information from multiple requests with guarantees about their order, you will need to chain promises by making subsequent requests *inside* the .then() calls of earlier requests.

Adding New Routes
Adding a new route is very simple and scalable. To do this, open the index.js file in the backend folder. This file contains all the routes that exist. Simply replace the route name and the database query for the route you want to return.

Adding a route with no parameters:

```
app.get('/[ROUTENAME]', (req, res, next) => {

    let queryStr = `
           [DATABASE QUERY]`
    pool.query(queryStr)
        .then(data => {
            console.log(data.rows);
            res.send(data.rows);
        })
})
```

Adding a route with parameters:

```
app.get('/[ROUTENAME]', (req, res, next) => {
    const parameters = req.query.county_name

    let queryStr = `
        [Database QUERY]`
    pool.query(queryStr,parameters)
        .then(data => {
            console.log(data.rows);
            res.send(data.rows);
        })
})
```

Suggestions for System Expansion
Currently, there is only one dropdown menu which changes the county selection for both the first and second line graphs. A different dropdown could be used for each graph to show data from different counties at the same time. If the developer wants to expand the project scope beyond the Lehigh Valley, they can modify the database and API to distinguish between Valley and non-Valley counties, then simply add whatever counties and colleges they want. The frontend app should not require any modifications to display more counties and colleges, unless new types of college or county aggregation are introduced.

Appendices

Database Tables

Counties
- <u>`name`</u>
- `population`

County Weekly Cases
- <u>`date`</u>
- `new_cases`
- `new_deaths`

Colleges
- <u>`name`</u>
- `fall_start_date`

College Weekly Cases
- <u>`date`</u>
- `new_cases`

Routes

Route	Information
http://139.147.9.191:80/counties	Method: GET Description: The response returns the name and population of each county Request: N/A Response: A list of JSON object with name and population of counties
http://139.147.9.191:80/colleges	Method: GET Description: The response returns the name, county name and fall start date of each college

	Request: N/A Response: A list of JSON object with name, county name and fall start date of colleges
http://139.147.9.191:80/countydata	Method: GET Description: The request sends in a county name in its body and the response returns the county name, date, new cases and death on a weekly basis for the county name in the body Request: county_name Response: A list of JSON objects with county name, date, new cases and new deaths.
http://139.147.9.191:80/collegedata	Method: GET Description: The request sends in a college name in its body and the response returns the college name, date and new on a weekly basis for the college name in the body Request: college_name Response: A list of JSON objects with college name, date and new cases.
http://139.147.9.191:80/allcountydata	Method: GET Description: The request sends in a county name in its body and the response returns the county name, date, new cases and death on a weekly basis for the county name in the body Request: N/A Response: A list of JSON objects with county name, date, new cases and new deaths.
http://139.147.9.191:80/countycolleges	Method: GET Description: The response returns the name, county name and fall start date of each college for all colleges in the county given in the request Request: county_name Response: A list of JSON object with name, county name and fall start date of colleges