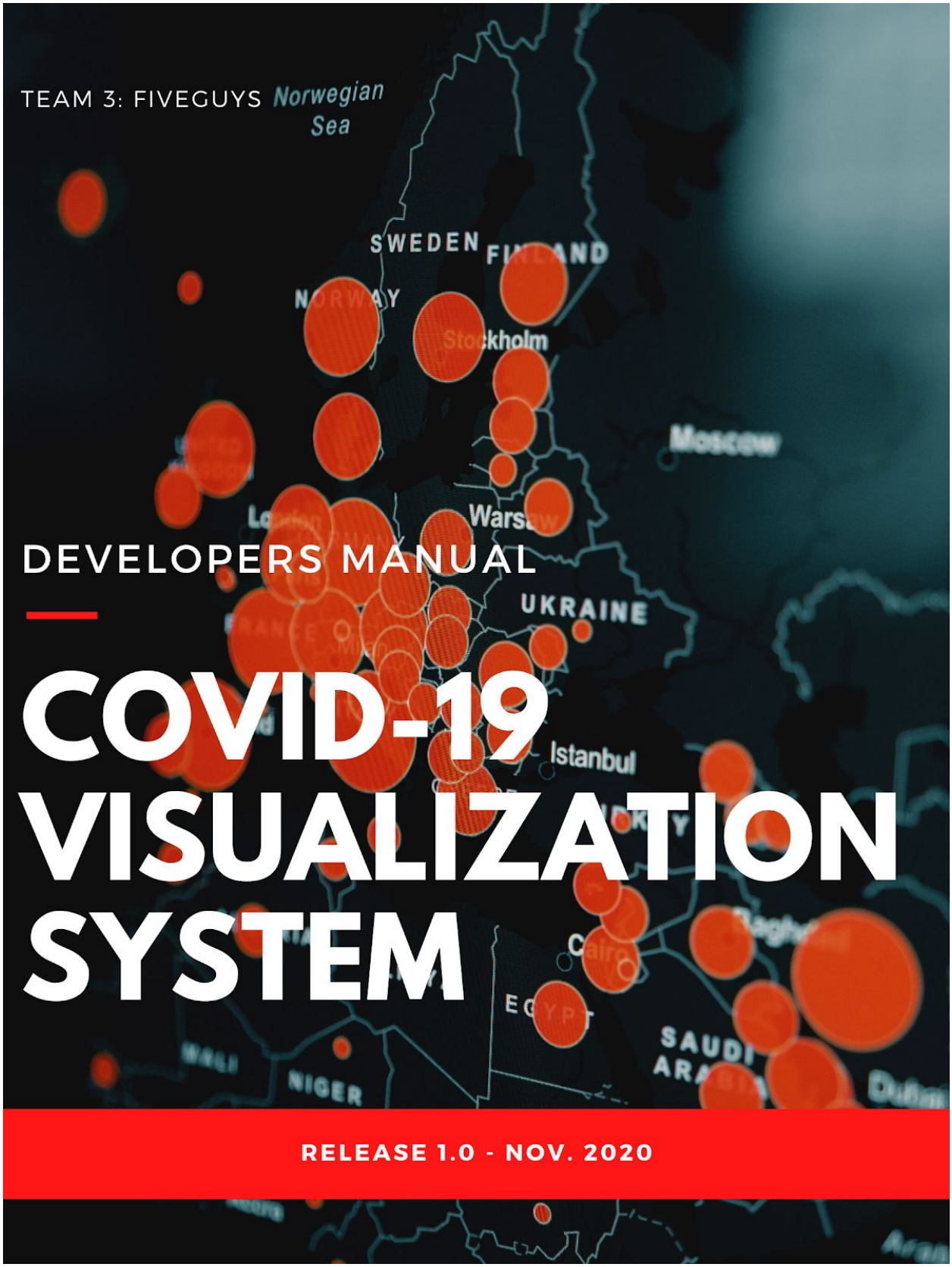

TEAM 3: FIVEGUYS
Norwegian Sea
SWEDEN
NORWAY
Stockholm
Moscow
Warsaw
UKRAINE
Istanbul
Cairo
EGYPT
SAUDI ARABIA
NIGER
DEVELOPERS MANUAL
COVID-19 VISUALIZATION SYSTEM
RELEASE 1.0 - NOV. 2020

Table Of Contents

Introduction to Stack

ApexCharts.js

ApexCharts.js is an open source Javascript charting library that aids developers to create beautiful and interactive charts for the web. These libraries are used to create the visualization charts for this project.

NPM & Node.js with Express

Node Package Manager (NPM) is a dependency management system that allows developers to import multiple Node.js libraries through command line inputs into project directories for their programs.

Node.js with Express serves as the web server application framework for this project. It enables us as developers to create API interfaces for us and other developers to communicate to the project's database to query COVID-19 data stored in the database.

PM2

PM2 is a production process manager for Javascript applications. It is used to start, pause, delete and monitor running Javascript application processes on a computer. This serves as a manager for our server application keeping logs of state of application, errors encountered during operation and mechanisms to start, restart and delete processes. It offers a change to run our application on the remote server in the background making it easy for us to focus on feature implementations.

PostgreSQL

PostgreSQL is the database managed and offered by the Computer Science department at Lafayette College. This is currently the database in use for the project.

Running the Application

Our application is hosted at http://139.147.9.198/ . To set up the program as we did, we would need one thing: A Lafayette College provided remote server with sudo permissions.

The fully implemented application can be reviewed and retrieved from the following link: https://github.com/rphan038/Lehigh-Valley-COVID19-Database-Project

Setup

First log on to remote server with username and password and disable the Apache server running on the server by running the command:

- `sudo systemctl stop apache2`

This stops the initial server running to stop so the Node.js with Express server can be installed.

Database

Database setup is already installed on the server. To access the one used for the project, enter the following commands into the command line:

- `psql`
- `\connect postgres`

These commands lead you to the space where all the tables used for the project are used. To see all the tables enter the command `\dt`. Check the database documentation in the appendix for a better understanding for the setup of the database.

To learn more about the commands and syntax of PostgreSQL, you can click the following link provided by the PostgreSQL developers: https://www.postgresql.org/docs/

Server

To set up the server we need to do a few things first:

1. Login into remote server and navigate to `/var/html/www/`
2. Remove all files from the directory using the `sudo rm <filename>` command
3. Install Node.js and NPM using commands:
    - `sudo apt-get update`
    - `sudo apt-get install nodejs`
    - `sudo apt-get install npm`
4. Get application from GitHub using command:
    - `git clone https://github.com/rphan038/Lehigh-Valley-COVID19-Database-Project.git`
5. Install all necessary packages in the `package.json` using command:
    - `npm install pm2`
    - `npm install express`
    - `npm install pg`
6. Find the file named `dbpg.js` and insert your Lafayette College remote server username and password using command:
    - `nano dbpg.js`
    - Find the following code in the file and follow the directions in the comments

```
const client = new Client({
     host: 'localhost',
     port: 5432,
     user: 'username', //Insert your server username here
     password: 'password', //Insert your server pw here
     database: 'postgres'
});
```

Process Manager

Since all required libraries are installed from the earlier commands. Running the server is very easy. Just enter:

- `pm2 start dbpg.js`
  This command starts the server running at port 80. To check status, enter:
- `pm2 status`
  To stop server, enter:
- `pm2 stop dbpg.js`
  To restart server, enter:
- `pm2 restart dbpg.js`
  More commands for managing process and handling errors can be found at:
- `https://pm2.keymetrics.io/docs/usage/pm2-doc-single-page/`

Extending the Application

Front-End

Adding UI Elements

The approach we took to design the UI was a focus on presenting the data and provide information and clarification for the user to be able to understand the COVID-19 data. Therefore, extra features we deemed unnecessary were not added.

The basic framework of the UI is (listed in order of top of the page to the bottom):

1. The title and welcoming message
2. Graph selection and county selection (if applicable)
3. Data visualization
4. Graph description

HTML, JavaScript, CSS, and BootStrap were used in this implementation. If you would like to add additional features, animations, descriptions, or styling, the UI files can be accessed

by navigating to the `/var/www/html/public` directory. From this directory, you can edit the `index.html`, `main.js`, and `style.css` files.

Data Visualization

As mentioned, ApexCharts.js was used to display the data. Primarily, line and bar column charts were used, but ApexCharts.js provides a wide array of graphs to visualize data as well as multiple features for each graph, which can be referenced at:
https://apexcharts.com/docs/installation/#

Editing existing graphs

1. Navigate to the `/var/www/html/public` directory
2. Enter `nano main.js`
3. Edit the `graph1()`, `graph2()`, `graph3()`, or `graph4()` functions according to what graph (descriptions for which graph each function corresponds to is indicated in the comments) you want to change and follow the ApexCharts syntax provided in their documentation to add your changes.

Adding new graphs

1. Navigate to the `/var/www/html/public` directory
2. Enter `nano index.html`
3. Edit code as follows:

```
<option>Select Graph...</option>
<option value="1">Cases and Deaths ...</option>
<option value="2">Incidence of Cases by County ...</option>
<option value="3">Cases by College in the Lehigh Valley</option>
<option value="4">Aggregated Cases of Colleges and ...</option>
                <!-- Add the following code -->
<option value=”5”> New Graph Title of Your Choice </option>
```

4. Next, the `main.js` needs to be edited. The general framework for how the different graphs are called is provided:

```
function graphData(){
     //Grab data from DOM
     let graphChoosen=document.getElementById('selectedGraph');
     let typeChoosen = graphChoosen.value;
     if(typeChoosen == 1) {
          //Call API endpoints, manipulate data
          graph1();
```

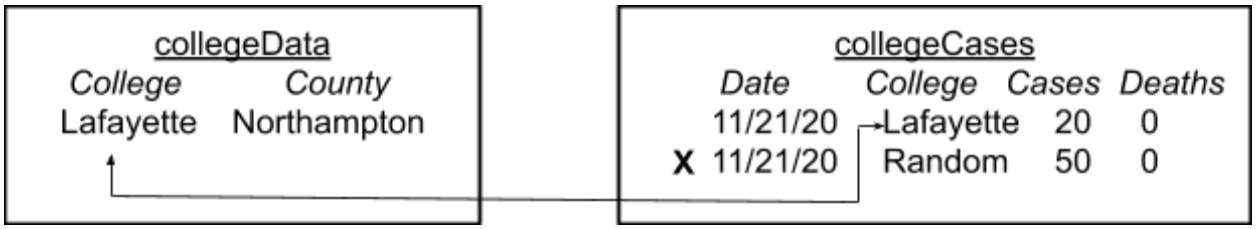
```
    } else if(typeChoosen == 2) {
    .
    .
    .
    } else if(typeChoosen == 4) {
        //Call API endpoints, manipulate data
        graph4();
    } else if(typeChoosen == 5) { //Add this if statement
        //This is where you can call different API endpoints
        graph5(); //You will need to create this new function
    }
}

function graph5() {
    //Follow the already implemented graphs in main.js
    //You can also reference the ApexCharts.js documentation
}
```

Back-End

Updating the Database

The first step in adding additional attributes or entries into the database is identifying what information is needed as well as what relations they are related to. Currently, there are four relations (countyData, countyCases, collegeData, and collegeCases), whose schemas can be referred to in the appendix. There are several foreign key constraints which specify that counties and colleges in countyCases and collegeCases respectively must exist in the countyData and collegeData relations.



As shown in the example above, for an entry to exist in collegeCases, the value of the "College" attribute in collegeCases must exist in collegeData. Since Lafayette exists in collegeData, the first entry of collegeCases is valid. There is no "Random" value in collegeData's College attribute, so the second entry of collegeCases cannot be added to the relation.

All other foreign key constraints are similar to the one just described and can be referenced in the database documentation in the appendix. Normal primary/unique keys are also used to prevent duplicate tuples and redundant information, which can also be referenced in the database documentation.

Please note that when entering in county cases, that the data must be the cumulative cases and deaths on the specified date.

RESTful API

In order to add additional endpoints to the RESTful API, it is best to familiarize yourself with the database schema, the NodeJS express module, and the existing API endpoints. This is because the API uses SQL queries to retrieve data from the database and displays that data to the user in a JSON format.

The general code for adding an additional endpoint through the above format is as follows:

```
app.get('/newEndpoint', function(req, result) {
     client.query('my SQL query', (err, res) => {
          if(err) {
               console.error(err);
               return;
          }
          result.status(200).send(res.rows);
     });
});
```

Fetch Data from Back-End

The list of existing endpoints that the application can retrieve JSON data from is located in the API documentation. There are multiple endpoints and parameters that will result in different data (see API documentation).

To use data after one call to the API use the following general code:

```
fetch('http://139.147.9.198/APIEndpoint/any/parameter', {
     method: 'GET',
     }).then(response => response.json())
          .then(data => {//use resulting data here });
```

There may be instances where you need data from multiple API calls. In this case, you would want all of the API call processes to finish before using the data.

```
Promise.all([
     fetch('http://139.147.9.198/APIcallOne'),
     fetch('http://139.147.9.198/APIcallTwo')])
     .then(function(responses) {
          return Promise.all(responses.map(function(response)
                                   {return response.json()}));
     }).then(function(data) {
          //data[0] contains data from the first API call
          //data[1] contains data from the second API call
     });
```

Limitations of Systems and Useful System Enhancements

A major limitation of our current setup is the manual command line insertion and update to our database. It makes it difficult to update the database, add new tables and delete data.

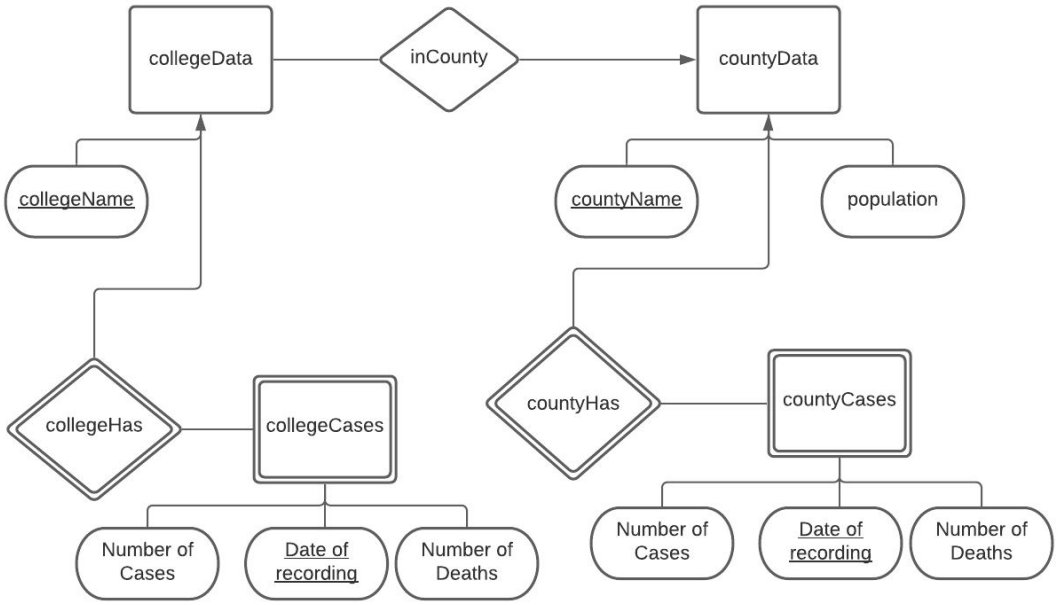
A solution to that could be an administrator page which requires login so an administrator could use the UI and make changes to the database without entering any command line input.

Another added advantage would be the automatic parsing of coronavirus data and update of the database when there is a change at the source for our coronavirus data. This would eliminate manual updates of the database.

Appendix

Database Documentation

1. **Explanation of E/R Diagram & Components**



Top-Level

Our E/R diagram explains on the top level how all the entities of the project relate to each other. We decided to break down the system into four major entity sets with their attributes. They are:

a. CollegeData
b. CountyData
c. CollegeCases
d. CountyCases

Entities

a. CollegeData

This entity set stores the basic information about a college. The only attribute for this entity set is the `collegeName` attribute which is considered the primary key for the entity. We assumed that no college in the Lehigh Valley has the same name hence our design decision.

b. CountyData

This entity set stores the basic information about a county. The two attributes for this entity are ‘population,’ which stores the population of a county, and ‘countyName,’ which stores the name of a county. The primary key for this entity is ‘countyName’

because we assume that no county will have the same name; whereas, there is a possibility that ‘population’ could be the same for a county.

c. CollegeCases

This entity set stores the basic information of recorded and death cases of coronavirus on college campuses. The attributes for this entity set are the `Date of Recording`, `Number of Cases` and `Number of Deaths`.

Our key is the `Date of Recording` and `CollegeName`. We assumed that no data will be collected twice for the same date or the date the data was recorded is an aggregate data for all recorded cases, hence the decision for it to be our primary key.

CollegeCases is a weak entity because it cannot be an independent entity. It is dependent on what college its data is referring to which is another entity.

d. CountyCases

This is a weak entity that keeps track of the case and death recordings in any particular county in the Lehigh Valley. This entity is weak because the data is dependent on what county the data is referring to, which is stored in another entity. The attributes that belong in this weak entity are ‘Number of Cases’, ‘Date of Recording’, and ‘Number of Deaths’. The key for this weak entity is the date of the recording and the county name.

Relationships

The relationships in this E/R diagram are:

a. inCounty

This relationship relates the collegeData and countyData entities together. This is necessary to determine the county name of which the college in question is located in. The key of this relationship is the `collegeName` as well as the `countyName` because they are the keys of the two respective entities we want to link together. Since there can be multiple colleges in one county and every county the college resides in must exist, there is a multiplicity of many-to-one from collegeData to countyData with a curved arrow pointing at countyData.
**Our diagram cannot show the referential integrity relationship due to the limitations of LucidChart.

b. collegeHas

This relationship is a weak relationship connecting CollegeData and CollegeCases with CollegeCases being the weak entity. This relationship has no attributes. The purpose of the relationship is to provide a trend of data that explains the coronavirus case count of colleges as time progresses. Keys are 'collegeName', collegeName' and 'Date of Recording'.

This relation is many-to-one from collegeData to CollegeCases. CollegeCases cannot be associated with more than one college and they must be associated with a college entity.

**Our diagram cannot show the referential integrity relationship due to the limitations of LucidChart.

c. countyHas

This weak relationship links the countyCases and countyData relations together so that if there are multiple recordings of cases/deaths in the countyCases entity, we can identify which county the data is referring to. The key for this weak relationship is 'Date of Recording', 'countyName', and 'countyName'. Additionally, this weak relationship has a multiplicity of many countyCases to one county where the arrow pointing at the countyData relation is curved because there must exist a county for the respective recorded data.

**Our diagram cannot show the referential integrity relationship due to the limitations of LucidChart.

2. Schema of Relations

a. pg_dump results

i. collegeData Relation

```
CREATE TABLE public.collegedata (
    collegename text NOT NULL,
    countyname text
);

ALTER TABLE ONLY public.collegedata
    ADD CONSTRAINT collegedata_pkey PRIMARY KEY (collegename);
```

ii. countyData Relation

```
CREATE TABLE public.countydata (
    countyname text NOT NULL,
```

```
    population integer
);

ALTER TABLE ONLY public.countydata
    ADD CONSTRAINT countydata_pkey PRIMARY KEY (countyname);
```

iii. collegeCases Relation

```
CREATE TABLE public.collegecases (
    dateofrecording date NOT NULL,
    collegename text NOT NULL,
    cases integer,
    deaths integer
);

ALTER TABLE ONLY public.collegecases
    ADD CONSTRAINT date_college_key PRIMARY KEY (dateofrecording,
collegename);

ALTER TABLE ONLY public.collegecases
    ADD CONSTRAINT collegecases_collegename_fkey FOREIGN KEY
(collegename) REFERENCES public.collegedata(collegename);
```

iv. countyCases Relation

```
CREATE TABLE public.countycases (
    dateofrecording date NOT NULL,
    countyname text NOT NULL,
    cases integer,
    deaths integer
);

ALTER TABLE ONLY public.countycases
    ADD CONSTRAINT date_county_key PRIMARY KEY (dateofrecording,
countyname);

ALTER TABLE ONLY public.countycases
    ADD CONSTRAINT countycases_countyname_fkey FOREIGN KEY
(countyname) REFERENCES public.countydata(countyname);
```

3. Data Dictionary

Data Dictionary				
Attribute Name	Attribute Description	Domain of Values	Attribute's Relations	Attribute Source(s)
collegeName	Name of college	Non-null Text value	collegeData	05:NULL,06:NULL,07:NULL,08:NULL,09:NULL,10:NULL,11:NULL
countyName	Name of county	Non-null Text value	countyData, collegeData	01:County Name,02:County Name,03:County Name,04:County Name
population	Population of county	[0 ,+2147483647) Integer	countyData	12:Population, 13:Population
dateOfRecording	Date of data point recording	Non-null date value	collegeCases, countyCases	05:NULL,06:NULL,07:NULL,08:NULL,09:NULL,10:NULL,11:NULL,01:NULL,02:NULL,03:NULL,04:NULL
numberOfCases	Number of cases for specific data point	[0 ,+2147483647) Integer	collegeCases, countyCases	05:NULL,06:NULL,07:NULL,08:NULL,09:NULL,10:NULL,11:NULL,01:NULL,02:NULL,03:NULL,04:NULL
numberOfDeaths	Number of deaths for	[0 ,+2147483647)	collegeCases, countyCases	05:NULL,06:NULL,07:NULL

	specific data point	Integer		,08:NULL,09:NULL,10:NULL,11:NULL,01:NULL,02:NULL,03:NULL,04:NULL

4. Data Provenance

Data Provenance			
Source Identifier	Source Name	Source Access	Source Location
01	Northampton County	This source contains all the data necessary for the county's COVID-19 deaths and cases over a period of time. This data can be found on the top two graphs of the page. Please see the next column for the source's location.	https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/state/pennsylvania/county/northampton-county
02	Warren County	This source contains all the data necessary for the county's COVID-19 deaths and cases over a period of time. This data can be found on the top two graphs of the page. Please see the next column for the source's location.	https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/state/new-jersey/county/warren-county
03	Lehigh County	This source contains all the data	https://usafacts.org/visualizations/coron

		necessary for the county's COVID-19 deaths and cases over a period of time. This data can be found on the top two graphs of the page. Please see the next column for the source's location.	avirus-covid-19-spread-map/state/pennsylvania/county/lehigh-county
04	Carbon County	This source contains all the data necessary for the county's COVID-19 deaths and cases over a period of time. This data can be found on the top two graphs of the page. Please see the next column for the source's location.	https://usafacts.org/visualizations/coronavirus-covid-19-spread-map/state/pennsylvania/county/carbon-county
05	Lafayette College	This source contains all the COVID-19 case and death data for Lafayette College. All necessary data can be found on the first two charts on the page. Please see	https://covid19.lafayette.edu/covid-19-dashboard/

		the next column for the source's location.	
06	Lehigh University	This source contains all the COVID-19 case and death data for Lehigh University. All necessary data can be found on the first chart and first table on the page. Please see the next column for the source's location.	https://coronavirus.lehigh.edu/covid-19-dashboard-reporting
07	Moravian College	This source contains all the COVID-19 case and death data for Moravian College. All necessary data can be found on the dashboard that appears. Please see the next column for the source's location.	https://www.moravian.edu/fall-2020/covid-dashboard
08	Muhlenberg College	This source contains all the COVID-19 case and death data for Muhlenberg College. All necessary data can be found on the dashboard that appears. Please see the next column for the source's location.	https://www.muhlenberg.edu/covid19/dashboard/index.php
09	DeSales University	This source contains all the	https://www.desales.edu/inside-desales/

		COVID-19 case and death data for DeSales University. All necessary data can be found on the dashboard that appears. Please see the next column for the source's location.	emergency-updates/covid-19-statistics
10	Cedar Crest College	This source contains all the COVID-19 case and death data for Cedar Crest College. All necessary data can be found in the first box that is on the page. Please see the next column for the source's location.	https://cedarcrest.edu/healthservices/covid.shtm#community
11	Pennsylvania State University Lehigh Valley	This source contains all the COVID-19 case and death data for Penn State Lehigh Valley. All necessary data can be found on the dashboard that appears and by selecting Lehigh Valley. Please see the next column for the source's location.	https://app.powerbi.com/view?r=eyJrIjoiMDFhMzI2YzQtNmQwNC00YjgzLWFjMzAtZmFlNGQyZGZiZGJhIiwidCI6IjdjZjQ4ZDQ1LTNkZGItNDM4OS1hOWMxLWMxMTU1MjZlYjUyZSIsImMiOjF9
12	County Populations (PA)	This source provides the populations for all counties in Pennsylvania. The data for each	https://www.pennsylvania-demographics.com/counties_by_population

		particular county can be procured by searching through the list to find the desired county and its respective population size. Please see the next column for the source's location.	
13	County Populations (NJ)	This source provides the populations for all counties in New Jersey. The data for each particular county can be procured by searching through the list to find the desired county and its respective population size. Please see the next column for the source's location.	https://www.newjersey-demographics.com/counties_by_population

API Documentation

Website Link: http://139.147.9.198/

Contents

County Data

HTTP Request: GET county/{countyname}

Comment: Retrieves a county's aggregate COVID-19 information on number of cases and death. Returned as a JSON array.

JSON Response:

```
[
  {
    "countyname":
    "population":
    "dateofrecording":
    "cases":
    "deaths":
  }
  ...
]
```

Path Parameters:

OPTION	**DESCRIPTION**
countyname	Name of county whose data needs to be queried. County Names available: Carbon, Warren, Northampton, Lehigh **(case sensitive)**

Response Attributes

KEY	**TYPE**	**DESCRIPTION**
countyname	String	name of the county
population	Integer	population of the county
dateofrecording	TimeStamp	Date the data point was recorded
cases	Integer	Cumulative cases up to said date
deaths	Integer	Cumulative deaths up to said date

College Data

HTTP Request: GET college/{collegename}

Comment: Retrieves a college's COVID-19 information on number of cases and death on recorded date stamps. Returned as a JSON array.

JSON Response:

```
[
```

```
 {
  “collegename”:
  “countyname”:
  “dateofrecording”:
  “cases”:
  “deaths”:
 }
 ...
]
```

Path Parameters:

OPTION	**DESCRIPTION**
collegename	Name of college whose data needs to be queried. College Names available: CedarCrest, DeSales, Lafayette, Lehigh, Moravian, Muhlenberg, PennsylvaniaState **(case sensitive)**

Response Attributes

KEY	**TYPE**	**DESCRIPTION**
collegename	String	name of the college
countyname	String	name of the college’s county
dateofrecording	TimeStamp	Date the data point was recorded
cases	Integer	Cases recorded on date associated with JSON object.
deaths	Integer	Death recorded on date associated with JSON object.

College Aggregated Cases and Deaths

HTTP Request: GET collegeagg/{collegename}

Comment: Retrieves all the college’s aggregate COVID-19 information on number of cases and death. Returned as a JSON object.

JSON Response:

```
{
 “collegename”:
```

“countyname”:
“dateofrecording”:
“cases”:
“deaths”:
}

Path Parameters:

OPTION	**DESCRIPTION**
collegename	Name of college whose data needs to be queried. College Names available: CedarCrest, DeSales, Lafayette, Lehigh, Moravian, Muhlenberg, PennsylvaniaState **(case sensitive)**

Response Attributes

KEY	**TYPE**	**DESCRIPTION**
collegename	String	name of the college
countyname	String	name of the college’s county
dateofrecording	TimeStamp	Date the data point was recorded
cases	Integer	Cumulative cases up to said date.
deaths	Integer	Cumulative deaths up to said date.

Total Number of Cases by Day in the Entire Lehigh Valley

HTTP Request: GET allcountycases/

Comment: Calculates the total number of recorded COVID-19 cases in the Lehigh Valley on every available day since March 2020

JSON Response:
{
“dateofrecording”:
“sum”:
}

Response Attributes

KEY	TYPE	DESCRIPTION
dateofrecording	TimeStamp	Date the data point was recorded
sum	Integer	The total number of cases recorded on said date