Technical Report

By Cameron Doggett, Jeffrey Moulckers,
Shrivu Shankar, Vassi Gianitsos, and Yash Shenvi Kakodkar

Summary & Motivations

The coronavirus has negatively impacted millions of people and hugely altered the world in which we live. Our team is interested in dissecting the different outcomes brought by contracting the coronavirus relative to the many locations the virus reaches. As a result, we built our website to serve as a database exposing different countries, potential risks, and case statistics surrounding the coronavirus. We hope that users realize the reality of the data we provide. We believe this site will inform users with relevant data that will hopefully motivate people to be careful about social distancing and spreading viruses in general. Our site allows users to categorize data based on country, making relevant statistics, such as total cases or total deaths, easily comparable across locations around the world.

User Stories

Customer

We were given 15 user stories by our customer and they are listed as follows:

Phase 1

1. **Map UI**
   We added a map of the whole world on the splash page and maps of either capital cities or the entire country in each of the instance pages. The estimated time for completion was around 5 hours as we needed to set up the Mapbox package before we could reuse it in various pages.

2. **Graphs**
   We added a graph that tracks the number of new cases daily in a country in the case statistics instance page. In future phases, we plan to add more informational graphs and style them better. Estimated time for completion was 2 hours.

3. **News page**
   The customer wanted a page where they can read articles about COVID 19 from around the world. To address this, we added a separate news page tab in the navbar which showcases different articles. Articles for a specific country can be found in that country's respective instance page.

4. **Info on testing**
   We added an attribute that can be in the response of our case-statistics API endpoint that contains testing information per country. The information was added in the case instance page. Since testing method data is difficult to provide due to various inaccuracies, we provide information on the total number of tests.

5. **Capital**
   We added the country's capital, a map of it and an image of it in each country's instance page.

Phase 2

6. **Ability to compare 2 countries**
   This user story is actually a feature that's optional in Phase 4 of the project, so we plan to implement it then so that we can better focus on Phase 2 deliverables. For right now, users can compare risk factors for a specific country to the global aggregate.

7. **Search bar functionality**
   This user story is actually a feature that's required in Phase 3 of the project, so we plan to implement it then so that we can better focus on Phase 2 deliverables. The total time to complete this was 3 hours.

8. **Threat level indicator**
   For the threat level indicator, we added a label describing the threat level as 'high' or 'medium' based on the country's daily case increase in the risk factors page. Its underlying determining factors will change as we understand the virus more. The total time to complete was 30 minutes.

9. **Clickable/interactive map**
   The map serves as an alternate way to navigate the website, and the global stats include statistics regarding the worldwide total number of cases, deaths,

recoveries, and active cases. This is featured on the main landing page. The total time to complete this was 5 hours.

10. **View global stats on the main page**
    We added an area to the main page that shows global stats that include the worldwide total number of cases, deaths, recoveries, and active cases**.** In future phases, we plan to better integrate and style these user stories (#9 and #10) for a cohesive user experience. The total time to complete this was 30 minutes.

Phase 3

11. **Number readability**
    We updated the format of our numbers so that users can easily read info. Numbers now have commas and the changes can be found on all pages. The total time to complete this was 1 hour.

12. **Pagination**
    We added pagination in earlier phases, but refined it so that the default items per page is at 10 for better viewing. We also added pagination buttons on the top and bottom for less scrolling. The total time to complete this was 1 hour.

13. **Sortable countries**
    Countries in the country page are sortable, as well as all other models in their respective model pages. They can be sorted alphabetically and also by other attributes unique to that model.

14. **Global News**
    The customer indicated that some of the articles were dated, but with the limitations of the free tier of NewsAPI, we decided to sort the news articles from newest to oldest so that users can focus on the newer info first.

15. **Tool tips**
    We included tooltips in the case statistics and risk factor instance pages. A user can now hover their cursor over the data they wish to see more info on, and a tooltip will appear.

Provider

We have assigned our provider 15 user stories that we thought were useful for anybody interested in learning about colleges in the United States. They can be summarized as follows:

Phase 1

1. **Finding colleges in a region within a range of tuitions**
   This would help students find colleges that fall in their budget and can help them make decisions about what they can afford.

2. **Ability to find colleges based on a particular field of interest**
   This would be useful to a potential applicant to see what they could do at that college.

3. **Comparison of schools relating to certain criteria**
   Being able to compare schools would help students decide which university to accept.

4. **View the options for financial aid at universities**
   This would help students determine what type of financial preparations they need to make should they choose to attend that university.

5. **Statistics, images, and data on universities presented in an appealing and organized manner**
   This would help the general user easily navigate the site and obtain the information they need quickly.

Phase 2

6. **An interactive map that shows universities by location**
   This would help to look at the surrounding area of a university and see where it's located.

7. **View information about alumni**
   This would be some fun, unique information to see which famous people attended a university.

8. **More statistics such as acceptance rates, average test scores and college rankings by majors**
   This would help students determine what they need to do to prepare to get accepted by that college.

9. **Table/grid based UI to quickly get an overview of all the content on the website**
   This would help to let the user quickly navigate the vast amount of colleges featured on the website.

10. **More visually appealing layout with graphics that presents the content cohesively with the overall website's theme**
    This would make the overall website look better and encourage users to come back and use the site.

Phase 3

11. **Make the column containing university names in the universities table wider**
    Some of the columns are oddly sized so it is hard to read relevant information.

12. **Make the links to the websites on the colleges page be hyperlinks which open in a new tab**
    It's difficult to navigate the complex website of a college in a small webview. Instead opening a new tab would make the navigation much easier.

13. **Have the navigation bar have distinct links to all three different models, rather than nesting them**
    This would help to make navigation much faster instead of having to click the dropdown menu each time you want to see a new model page.

14. **Present restaurant locations on a map**
    This would give a good idea regarding the food options around a campus and what's available.

15. **Ability to filter colleges by state**
    This would help with selecting and finding colleges based on a specific criteria set by the user.

Our REST API

Our RESTful API was designed first-and-foremost to meet the needs of our frontend application. The demands of the frontend portion of this project necessitated the creation of six endpoints, which are shown in the table below. Each model was assigned two endpoints that respond to GET requests in order to reflect the demands to retrieve all instances of that model and another to retrieve a single instance of the model by a unique country identifier. The former is very useful for presenting all instances of that model on a single page, such as the model pages with tables/grids, whilst the latter saves on server bandwidth and helps meet the need to display information on a single instance of that model, such as on the individual instance pages. Across all of these endpoints, an optional attribute query parameter enables the frontend client to specify exactly which fields they require in the response. This simultaneously lowers server bandwidth and serves to specify exactly which attributes the client can expect in the response. In phase two, we have added two additional endpoints to help with user-requested features on the frontend. The additions include: global news and global statistics. In phase three, we added the search endpoint, which enables the client to perform a search with a given query. Unfortunately, because a significant portion of our dataset is numeric, meaningful searches are primarily limited to country related textual data such as capital city, region, name, etc. The search endpoint responds with a list of country identifiers related to the given query.

API Endpoints Table

Endpoint	HTTP Method	Description	Path Parameters	Query Parameters
/countries	GET	Get all countries	N/A	attributes
/countries/:identifier	GET	Get a country	identifier	attributes
/case-statistics	GET	Get all instances of case statistics	N/A	attributes
/case-statistics/:identifier	GET	Get an instance of case statistics	identifier	attributes
/risk-factor-statistics	GET	Get all instances of risk factor statistics	N/A	attributes
/risk-factor-statistics/:identifi	GET	Get an instance of risk	identifier	attributes

er		factor statistics		
/global-news	GET	Get global news relating to COVID-19	N/A	N/A
/global-stats	GET	Get global statistics for COVID-19	N/A	N/A
/search	GET	Perform a search with a given query	N/A	query

The second goal we had in mind while designing the API was to allow any external client to easily interface and retrieve data from our database. To that end, our API does not currently require any kind of API key in order to make requests. Additionally, we provide thorough documentation of the COVID-19 DB API on Postman (see tools section for more details) and our project's repository, so that anyone can learn to make requests to our API.

Documentation for our API may be found at:

- https://documenter.getpostman.com/view/12799044/TVKJxuP4
- https://gitlab.com/jrmoulckers/covid19db-net/-/blob/master/backend/docs/documentation.md (Includes lots of details regarding available attributes per model and valid identifiers)

Our Models

Each model targets a specific grouping of data and statistics to localize information for a user to their specific country. This data is essential in understanding trends on a country to country basis and providing context as to why certain states have handled the pandemic more effectively than others.

Countries

As COVID-19 is a global issue, we decided to create a model representing each country to effectively group statistics for local reference.

Each country instance contains the following attributes and sub-attributes:

- Name

- Codes
    - ISO 3166-1 alpha-3 code
    - ISO 3166-1 alpha-2 code
- Calling codes
- Capital
    - Image
    - Location
        - Latitude
        - Longitude
    - Name
- Alternate names
- Region
    - Region
    - Subregion
- Population
- Location
    - Latitude
    - Longitude
- Area
- Time zones
- Borders
- Currencies
- Languages
- Flag
- Regional blocs
- News
- Sources

Each country instance has a 1:1 relationship with that country's corresponding case statistics and risk factors.

Case Statistics

Our case statistics include quantitative information specific to a country including total and active cases, tests, deaths, and recoveries. We maintain updated statistics by presenting their current values and comparing them to those of yesterday. Along with day to day statistics we also display the rate of change of many factors and present the country's case totals in an interactive graph.

Each case statistics instance contains the following attributes:

- Country
    - Name
    - Codes
        - ISO 3166-1 alpha-3 code
        - ISO 3166-1 alpha-2 code
- Date
- Location
    - Latitude
    - Longitude
- Totals
    - Cases
    - Deaths
    - Recovered
    - Active
- New
    - Cases
    - Deaths
    - Recovered
    - Active
- Smoothed new
    - Cases
    - Deaths
- Percentages
    - Fatality
    - Infected
    - Have recovered
    - Active
- Derivative new
    - Cases
    - Deaths
    - Recovered
    - Active
- History
- Sources
- Testing

    - New tests
    - New tests smoothed
    - Positive rate
    - Total tests

Risk Factors

Risk factor statistics represent some of the health and human development-related factors associated with a country and may be utilized to indicate if portions of the country's population are more susceptible to a fatal case of COVID-19 or if the country is less prepared to handle large outbreaks.

Each risk factor statistics instance contains the following attributes:

- Country
    - Name
    - Codes
        - ISO 3166-1 alpha-3 code
        - ISO 3166-1 alpha-2 code
- Location
    - Latitude
    - Longitude
- Population density
- Median age
- Aged 65 or older
- Aged 70 or older
- GDP per capita
- GINI
- Extreme poverty rate
- Cardiovascular death rate
- Diabetes prevalence
- Female smokers
- Male smokers
- Hospital beds per thousand
- Life expectancy
- Human development index
- Handwashing facilities
- Sources

For more details regarding any of the above models' attributes, please view the documentation.md file available in the RESTful API section.

Tools Used

Postman

In the first phase, Postman was used to generate an API documentation collection and to make requests to our API data sources for scraping. Documentation was created on Postman by defining an OpenAPI specification which outlines our API design and then generating a collection that corresponds to that definition. The API specification is named api-spec.json in our public repository. Second, we made GET requests to the API data sources from Postman and saved the responses to be used for our model instances.

In the second and third phases, we improved the OpenAPI specification to reflect new endpoints and additional attributes for each model. Using that refined specification, we regenerated the Postman documentation collection for our API and generated a new collection used for testing with Newman (see the testing section for more).

React

We use Facebook's React UI library for all of the UI shown on the site. The framework allows us to build modular and composable components that can be adjusted and adapted to different parts of the site. We write components in JSX (a mixture of JavaScript and inline HTML) which are then webpacked into JavaScript bundles and served to the user.

AntDesign

We utilize the pre-made and dynamic components provided by the AntDesign React UI library to create a more standardized interface and aesthetic across the site. Many of our most important elements, including the tables, grids, and buttons, are backed by this library in our implementation.

React-Bootstrap

We originally used this framework to get a quick and light static implementation up, but we have since shifted to primarily AntDesign components in the second phase. We will migrate fully from React-Bootstrap in future phases as we focus more on styling and UI/UX across our site.

React-Highlight-Words

We use this component to assist in highlighting search terms across the site. We created a customized component utilizing the key functionality of the Highlighter component from this plugin to concisely display filtered search results, highlighted with matching text across data in each model.

React-Router

We use this navigational component library built for React to enable navigation across the pages of our site. This allows dynamic declarative routing across models and enables instances to be accessible via links on the site, all integrated into React.

Mapbox

As our site relies heavily on locations and countries to display data from our models, we require a dynamic toolkit to visualize various components on and interact with a world map. The Mapbox API is incorporated into our site and used in our splash page to direct users to countries interactively, as well as in our model instances to display a visual location of the specific country.

GitLab

All of our code, including both frontend and backend, is hosted in a public repository on GitLab. GitLab enables us as developers to collaborate efficiently because it has a wide range of useful features including: an issue tracker, support for tracking a multitude of separate development branches, continuous integration, and more. We will continue to develop within the same repository throughout the lifecycle of the project.

Flask

We chose to develop our backend API with the minimal Flask framework over Django. It provides all of the basic necessities that we required to achieve our goals and has several very useful extensions, such as Flask-RESTful and Flask-SQLAlchemy, that made our jobs a lot easier.

Flask-RESTful

Flask-RESTful is an extension for Flask that enables us to easily define endpoints for our API. With Flask-RESTful, we define several Resources (classes) and attach them to routes in a succinct manner that is much more maintainable and readable than the original way of defining routes in Flask with function decorators.

Flask-SQLAlchemy

Flask-SQLAlchemy is also an extension for Flask like Flask-RESTful. With it, we define several model classes and their static properties, which are then transformed into tables in the database. Additionally, the module provides the ability to easily query entries in the tables without ever having to work with SQL. In doing so, we mostly avoid having to directly interface with SQLAlchemy, which has a rather steep learning curve.

SQLAlchemy

While we do not directly interface with SQLAlchemy in our code, we do so indirectly through Flask-SQLAlchemy, which is built on top of SQLAlchemy. This connects our backend to our database, from which we can query entries in each table.

SQLAlchemy-Searchable

We utilized the SQLAlchemy-Searchable module to help with servicing the requests to the API's search endpoint. The module provides full text search capability on textual column data, which is almost exclusively within the country model, and easily integrated with Flask-SQLAlchemy.

Docker

We utilized Docker to create separate images for both our frontend and backend development. The images enable us to easily and quickly set up a working development environment for all of our team members.

PostgreSQL

PostgreSQL was selected for our relational database because it is the premiere open source database at the moment and has many unique features, such as support for storing arrays and JSON data. Our database is hosted on a remote server and is independent of our backend server.

Python's unittest

Python's provided unittest library was used to create additional unit tests for the backend. Being both native to Python and easy to use made it an easy choice for this purpose.

DB Diagram

The DB Diagram tool enabled us to easily depict the structure and relationships of our database tables. The generated diagram for our database can be found in the diagrams directory of our repository.

Testing

Selenium

We created acceptance tests for GUI using Selenium, a tool used to automate web-based interactions in Python. The test file can be found in frontend/gui_tests.py and essentially contains tests that navigate through different parts of the website and check their content. In order to do this, the first step was to install a webdriver (in our case, one for Chrome) which would serve as the vehicle in performing the tests. Currently, the tests check each link in the navbar making sure they exist, interact with the sorting aspect of the tables found in the Cases and Risks pages, and look at individual instances of each model.

Jest

We created unit tests of the JavaScript code using Jest - a simple testing framework. In the test file, frontend/__tests__/tests.test.js, we used Enzyme, a testing tool which simplified rendering the components and functions we test throughout the project. Using the shallow() function from Enzyme constrained us to test each component as a single unit, only recognizing code defined within that component. We also used Jest to take snapshots of the react components and compare them to the newly rendered output with each test run.

Postman

As mentioned in the related section under tools, we refined our OpenAPI specification and generated a new Postman collection dedicated to testing the endpoints of our RESTful API. To run the Postman tests, we exported the collection to a JSON file which can then be run with Newman. The exported collection is available in our repository and is named postman.tests.json.

Python Unit Tests

In order to test some of the other functions we defined in our backend, we leveraged Python's unittest to define unit tests. Tests cover helper functions as well as some of the

methods used to interact with our database. The tests are located in the backend directory of our repository in the file named test.py and the testing module.

Database

We used Heroku (which uses AWS under-the-hood) to provision our PostgreSQL database due to it having an easy-to-use free tier. Our backend API service connects directly to the database as a root postgres user to query and format content. As mentioned before, the backend code primarily interacts with the database through Flask-SQLAlchemy. The design of the database and its tables, which represent the models, is specified by the use of Flask-SQLAlchemy's Model class and static data members of type Column. While we do specify an integer primary key for each table, we also include relevant country names and codes within each table as the intended, friendlier means of selecting from the tables. These country identifiers are all specified as unique and non-nullable. Additionally, while we did want to make the structure of the tables and their columns parallel the format of the data that is serviced by the API, we did have to make slight adjustments to the manner in which we store certain data, such as capital name and locations, if we wanted to easily read it from the database.

Hosting

We chose the Google Cloud Platform (GCP) for hosting because of its developer friendliness and available credits. Within GCP, we use two separate App Engine services for serving static web packed frontend assets and for running our Flask-based REST API. App Engine handles SSL certificate generation, orchestration, and deployment for us. We are using GoDaddy for managing our domain's nameservers and have a setup A, AAAA, and CNAME directives for routing traffic to GCP machines. App Engine's dispatching feature is used to direct this traffic to either the API or frontend service based on a set of pattern matching rules.

Data Organization

Search

Several aspects of our project involve search. In terms of the backend, a new search endpoint was implemented, which takes a given query parameter and responds with a list of relevant country identifiers. The implementation of this backend search functionality was accomplished with the full text search feature available in PostgreSQL,

using a library called SQLAlchemy-Searchable (see tools section). With the library, we specified a textual search vector in the Countries database model, composed of the country name, ISO codes, region and subregion, and capital city name, in order to enable the search functionality. The search endpoint was limited to the textual columns of the Countries model because the majority of our dataset is numeric, which textual search vectors cannot include. The sitewide search implemented the new search endpoint by utilizing a table from AntDesign to render all the new data. It features all pages in the website and each of the different models for each instance of a country.

Similar to the sitewide search, the model search for case statistics and risk factor statistics utilized a table to render the results as well. In this case however, the implementations each used the model's respective API endpoint instead of the search endpoint to populate the table. Afterwards, to add the search functionality directly on the tables themselves, we constantly take the current query in the search bar and use it to filter matches where table data attributes include the current query. By doing this we are able to also search by all columns found in the table. The results update automatically as you type since we constantly update the state of the underlying table data and query. The sitewide search table utilizes a similar automatic result feature as well.

The model search for countries required a slightly different implementation since the countries and their data are organized differently upon collection from the backend. When rendered on the frontend, we lose access to the simple search functionality provided across AntDesign tables, so searching is implemented in a unique way to query across certain data points and update visually in real time. The country card data is currently sorted, filtered, sliced upon each render update, and is then transformed to its visual representation. When a search query is entered, the components will perform this process upon each change, which we realize is currently fairly exhaustive, and leads to lag when users enter or backspace search characters rapidly. We plan to implement a change to this process in Phase 4 to follow a similar structure to the sitewide search and perform minimal operations to the data on each change to allow for a smoother experience. For now, though, all visible country data and text is searchable within the model.

Filtering

For each model we handled filtering based on the key attributes displayed in the model view of all instances. Filtering was fairly simple within AntDesign tables used in the case statistic and risk factor models, as we chose major groupings in which to cluster the attributes of each instance. In the country model view we had to define our own method

of filtering, which ended up handling sorting as a byproduct. A user is able to define which field to filter, as well as the range in which to filter and sort - either alphabetically or numerically.

Sorting

In sorting our models we found similar ease in the case statistic and risk factor models using our AntDesign tables, and we utilized the filter functionality with countries to incorporate a natural order based on the range defined by the user. Within the countries model a user is able to switch the upper and lower bounds to reverse the sorting order of the already filtered countries.

Pagination

For each model we added pagination from the AntDesign components library to allow for easier readability and selection of countries, case statistics, and risk factors. We allow control over the number of instances per page, and allow users to skip to pages or click through using the right and left arrows. In the future our models will allow for searching and sorting between instances so that a user does not need to solely base their search on the pagination tool. In addition, users may view the number of instances currently in view and may jump to a page via a concise page search bar.