Daniel Carson, Aaron Renner

Professor Liao

CIS485

29 October 2020

Interacting with RESTful APIs using a Blazor Application

Keywords: JSON, RESTful API's, Graphical User Interfaces, COVID-19, Coronavirus

1. Introduction

Before the start of the year 2020 only a few people in the world knew of the existence of the SARS-CoV-2 virus. As this deadly virus sweeps across the world there must be a degree of adaptations, we can take to progress the knowledge we have so that we come out victors of this pandemic. To tackle an enemy of this size head on the must be some foundation in place to help excel us onto the playing field, which is not a problem after the immense amounts of technology that is going into place to help fight this pandemic. The technology in question that is going to help this team create our project revolves around the massive amounts of public data that many companies are storing right now inside of databases which are connected through the internet. We believe the best source for finding more ways technology is directly guiding this pandemic should be accessed from "Digital technology and COVID-19" an article found on nature.com by Daniel Ting. As a team we are very motivated by the connected environment the word is moving at, we believe staying in the front line and finding ways to integrate a whole new connected medium through an existing portal is the future of connected devices.

Currently there is a war going on against an invisible enemy, the novel Coronavirus is having people relying on technology more than we could have ever predicted. The surge in need for better infrastructure to track the data for the coronavirus has led to many startup RESTful web services to power the delivery of these datasets. As developers we believe there is a need to connect these services through a multi-faceted application to the people for convenient access. This is where the second part of our study will focus, we will be delivering a Graphical User Interface that we will provide curated information after collecting some simple sorting information like State or Zip code.

The introduction of contact tracing applications has been a popular move by some countries to precisely track the movements of people through the phone, this can allow healthcare to better track the contact of someone who could be a carrier of the virus. These types of features and data collection could be integrated into any mobile application, hidden in the source code, we will not be covering this topic, but more can be found in the referenced article "COVID-19 Contact Tracing and Data Protection Can Go Together." Our application will focus on the data that one would be derived from this sort of first-party service and in particular we will be looking at the data from a RESTful service, this service would operate using the basic CRUD methods and communicate using a JSON data format type. Our team will be considering some of the application programming interfaces listed by Wendell Santos in his blog post "30 APIs to track COVID-19."

2. Literature Review

2.1 Designing a Framework with Test-Driven Development

One of our pieces of literature includes an article published by *IEEE* which supports and explains Test-Driven development. Test-Driven development is related to DevOps practices in the fact that it allows for more frequent and reliable software releases. For this project, we practiced Test-Driven development to ensure that we build one software component at a time and that each software component is reliable. In addition, Test-Driven development requires less regression testing since each software addition is tested right after it is added. Overall, Test-Driven development allows for quicker and more reliable software additions which was imperative for our project given the short deadline.

The author, Eduardo Guerra, a computer science researcher from Brazil, defines Test Driven Development as, “a development and design technique that creates tests before production code is created” (Guerra 9). A clear benefit of creating unit tests prior to writing code is that an expectation is set, or a test, which will fail until the code is implemented to meet the conditions of said test. Today, many organizations are following Test-Driven development practices as well as DevOps practices. For example, Daniel works for *American Public Education Inc.* and this industry requires their developers to create unit tests which guarantee requirements are met prior to making code changes. In our project, we decided to implement these practices to ensure that our code meets the expectations and requirements we set prior to writing the code.

In addition to Test-Driven development, this article also mentions the importance of refactoring code. The author mentions that this of development involves writing the simplest code possible to meet existing test cases (Guerra 13). As a result, as more complex test cases are added, the original methods that passed unit testing before may need to be refactored to meet the new test cases. While refactoring code, it is crucial to strictly build on the initial implementation while maintaining functionality that continues to pass old test cases. Furthermore, it is important to consider that that changing a method that is interconnected with others may cause several test cases to fail. Lastly, the author stresses that refactoring code is inevitable when following Test-Driven development practices and that it does not necessarily imply that mistakes were made (Guerra 13). As a team, we found this advice to be relevant since our first project baseline was to make a single API call and somehow display this information on-screen. Throughout our project, we refactored our code several times to remove repeated code, add new pages, optimize our application and add new functionality such as form data entry.

2.2 An Analysis of Public REST Web Service APIs

The largest part about the project we have in mind has to do with the connection of applications through a connected medium, the internet and more specifically a type of web service built to deliver object-oriented communication over the Hypertext Transfer Protocol denoted HTTP. The

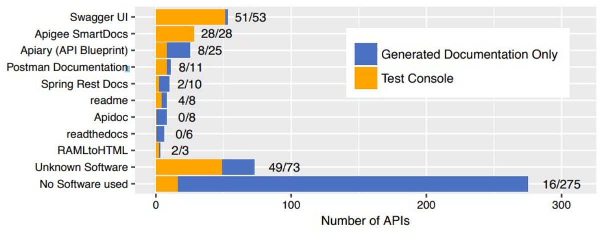


Fig. 1. Documentation tools used by the APIs

publication I speak of was in an IEEE Computer Society study done on 500 of the 4,000 most popular REST web services found on alexa.com and selected by the authors of the article Neumann, and Bernardino. The study started with the goals of identifying of best practices and providing URI examinations that are common, the study also lead to the analysis of the feature set provided by the services and comparing with the documented examples to check for validity. Most notably in the study’s REST architecture category they looked at the number for validity.

Most notably in the study's REST architecture category they looked at the number of unique request endpoints where located in the web service, these finds revealed that roughly 56% of the studied API's had at most 20 operations, one third provided between 2 and 10 operations and 5.2% provided more than 200 operations. Furthermore, revolving around the architecture of RESTful services there are similar features to API's that are not represented in similar ways, some of these features in question like output format selection, and API versioning selection. One interest point found by this study shown in the above figure, where you see the most popular documentation service is Swagger, which is not something we can cover in this report.

The things our team hopes to gain from the understating of this article is the practices and things to look at when we are using different Application Programming Interfaces throughout our project. We observe that the lack of standards among the different service available on the web could lead to some problems if we try to interact with the API's without looking into the documentation, if any is available.

2.3 A Web Service Based on RESTful API's and JSON

Starting at the very beginning of our project we needed required a reliable internet resource, this was a RESTful client which can be best described as a way to communicate with a text only website. The researches from this paper help us understand how these web services operate by providing the figure below to visually track how the data might get processed inside a proper API service. A RESTful web service is typically designed like an MVC that will accept data through its controller interface to be processed making sure proper data is inserted. After a RESTful client performs the pre-processing on the request it can be passed along to a service that will be able to process together a response based upon the requested information. Throughout our implementation we are only allowed to access the controllers through the URL, further implementations exist in a RESTful client in some cases including a full CRUD instruction set.

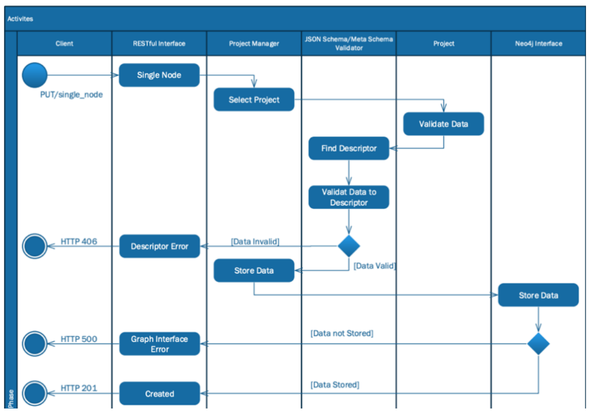


Fig 2 This flow represents the flow of data across a RESTful service. Used from the Resource references.

Through the proper communication with a RESTful interface you will receive this JavaScript Object Notion formation data in a communication packet that can be processed in the language of the data packet acceptor. The communication protocol between the server and our client happens through HTTP packets which can be processed all modern languages using various network connects(My Reference)[1] (https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=netcore-3.1) where the body of the HTTP packet is the JSON data, typically just one long sting. The more data which is composed by the RESTful service will linearly increase the response time unless able to be further optimized by the developers to be multi-threaded or some further compression can be done by the host.

3. Overview

The creation of an application which can show off the vision the team has for a Coronavirus information delivery will be achieved by utilizing basic functionality of data representation and how to acquire that data. On the internet there is a lot more than the beautiful visual instances of a web page, for instance the use of a popular language called JavaScript Object Notation or JSON for short was made for plaintext data transfer. To deliver this JSON data to webpages we look at a delivery method referred to as a RESTful web service, which is just a set of instructions provided to interact with some functionality of the web service. In most cases when interacting with a RESTful web service you will be using common JSON language as either a request body or it will be used as a response from the web service.

The structure of our application uses a C# backend where one might image the typical JavaScript language to be working, this is on purpose because of the use of a Blazor inside of the ASP.NET Core provided by Microsoft. While the data transformations happening throughout our backend in C# must not be forgotten that we are delivering everything in HTML for the User Interfaces. The Blazor interpreter allows us to receive the JSON data from a RESTful web service and perform a process called deserialization to transform it into simple C# objects. Please continue through this overview carefully to follow along with many of the resources we use throughout our application implementation.

3.1 The COVID Tracking Project

Throughout our application we discuss a RESTful web service for which we gather our data from and this is done all through the same service, a website called covidtracking.com is our trusted website of choice. The data supplied through their API's are said to be "taken directly from the websites of state/territory public health authorities." (https://covidtracking.com/about-data/faq#where-do-you-get-your-data) which is collected by volunteers and entered into the service manually. The API for this webpage allows us to gather a wide variety of data on the United States for example we can find any statistics for an individual date for the combined US or an individual State. We utilize this resource to display tables in our HTML Demo application on data for individual States in the US to specific data for a state over a date span. Follow the implementation for more information on how to get this running and try for yourself.

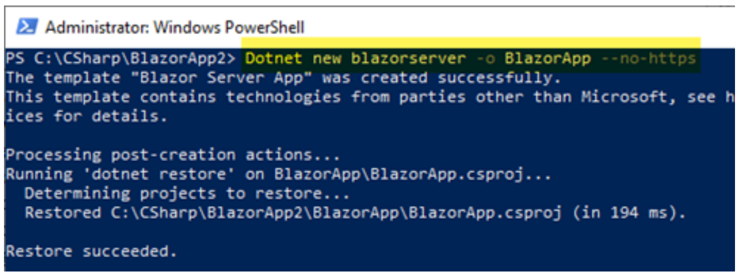
3.2 Overview of Blazor

The Blazor Webassembly, created by *Microsoft*, is one of the latest and greatest solutions for creating a web application that is responsive and dynamic. The Blazor Webassembly runs on .NET Core version 3.1 which means that developers using this method will need to be familiar with Windows Programming. One of the biggest appeals to a Blazor app is that backend developers, who may not be as familiar with web development, are now able to code web pages with minimal knowledge of JavaScript and HTML. Our team consists of both backend developers who are familiar with coding in Java and C-Sharp making the Blazor Webassembly the perfect candidate for our web app.

In addition, developing a Blazor Webassembly is less difficult for developers who are not familiar with web development because standard templates are available from *Microsoft* which make the job much easier. These templates demonstrate valuable examples such as contacting a Restful API or service to retrieve data to be displayed on-screen. Consequently, programmers without any knowledge of a Blazor web app are able to hit the ground running with a template that can be easily modified to meet their software component goals. Overall, Blazor apps are becoming quite popular in the industry since it allows backend developers with minimal knowledge of JavaScript to code a dynamic web app that pulls data from a REST API for example.

3.3 Creating/Testing your first Blazor app

As mentioned previously, we are backend developers and decided to code our web application using primarily the .NET framework. After some thought and research, we decided to code our application using *Microsoft's* latest and greatest web app ASP.NET or to use the latest and greatest technology, a Blazor Web Assembly. *Microsoft* provides an easy to follow 5-10-minute tutorial on how to start your first Blazor app. Prior to starting a Blazor App, you must first download .NET Core SDK 3.1.4 which provides the packages required for this technology (Blazor Tutorial). Once installed, create your first Blazor app by running the following command in Windows command prompt or PowerShell:



```
PS C:\CSharp\BlazorApp2> Dotnet new blazorserver -o BlazorApp --no-https
The template "Blazor Server App" was created successfully.
This template contains technologies from parties other than Microsoft, see h
ices for details.

Processing post-creation actions...
Running 'dotnet restore' on BlazorApp\BlazorApp.csproj...
  Determining projects to restore...
  Restored C:\CSharp\BlazorApp2\BlazorApp\BlazorApp.csproj (in 194 ms).

Restore succeeded.
```

Fig. 3. Command for creating a new Blazor app based on a template

If this command succeeded, a template Blazor app was successfully created in a new directory "./BlazorApp". Next change into the directory and run **dotnet run** to start the server which can be accessed at localhost:5000.

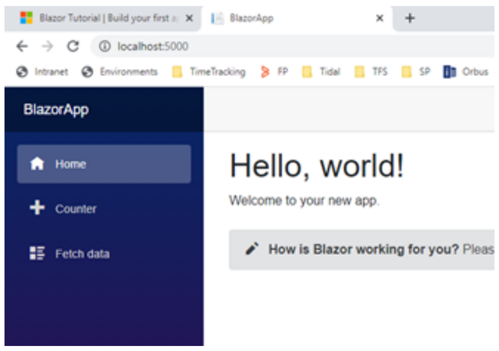


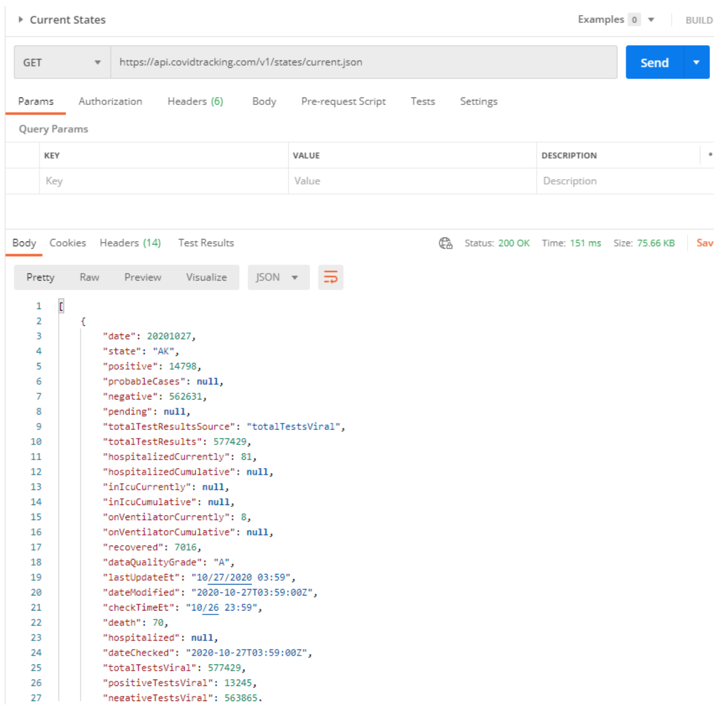
Fig. 3. Screenshot of Blazor Application running

4. Implementation

Commented [AR1]: 1.Background
2.Design
3.Implementation
4.Platform
5.Packages Used
Coding razor pages
Making Restful API calls using C#
CSS Section/Bootstrap
Docker? ?

4.1 Evaluating API calls

In order to analyze and evaluate the API endpoints provided by *Covidtracking.com,* we used an API development and collaboration tool called *Postman*. *Postman* allows even novice developers to get their feet wet with Restful API interaction. For our project, we used this application primarily to evaluate the reliability, availability and consistency of the data provided by *Covidtracking.com.* In addition, after testing and evaluating the API calls available from this source, we referenced the API documentation in cohesion with JSON output of the calls in *Postman* to design data models such as StateData and USData. In order to translate the JSON output into usable programming objects, the object models must include parameter names and data types that match with those found in the JSON output from a given API call. Testing an API call in *Postman* is as simple as creating a new request, entering the endpoint and clicking the "Send" button. (See figure below)



GET https://api.covidtracking.com/v1/states/current.json

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Status: 200 OK Time: 151 ms Size: 75.66 KB

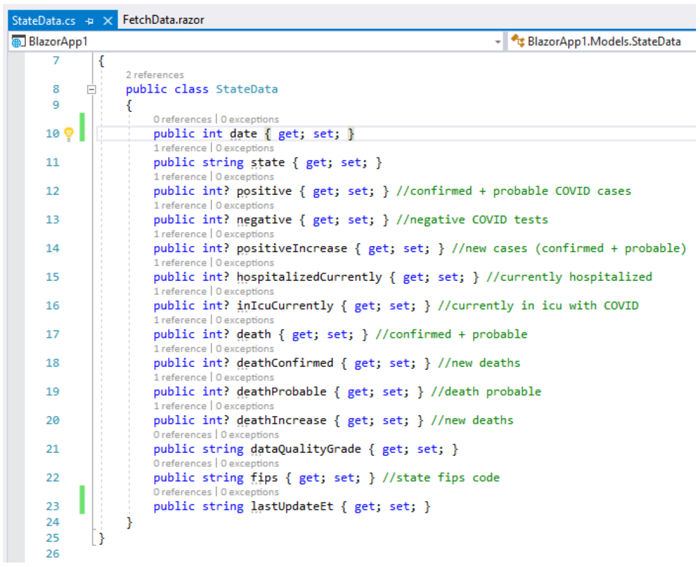
```
[
    {
        "date": 20201027,
        "state": "AK",
        "positive": 14798,
        "probableCases": null,
        "negative": 562631,
        "pending": null,
        "totalTestResultsSource": "totalTestsViral",
        "totalTestResults": 577429,
        "hospitalizedCurrently": 81,
        "hospitalizedCumulative": null,
        "inIcuCurrently": null,
        "inIcuCumulative": null,
        "onVentilatorCurrently": 8,
        "onVentilatorCumulative": null,
        "recovered": 7016,
        "dataQualityGrade": "A",
        "lastUpdateEt": "10/27/2020 03:59",
        "dateModified": "2020-10-27T03:59:00Z",
        "checkTimeEt": "10/26 23:59",
        "death": 70,
        "hospitalized": null,
        "dateChecked": "2020-10-27T03:59:00Z",
        "totalTestsViral": 577429,
        "positiveTestsViral": 13245,
        "negativeTestsViral": 563865,
```

Fig. 4. Making an API call in Postman

4.2 Creating Data Model for API Reponses

As discussed in section 3.3, we followed a *Microsoft* tutorial to create our first Blazor application. Next, we worked collaboratively to add our first API call our new Blazor web application. As a proof of concept for our Capstone Proposal, we practiced making a RESTful

API call in a Windows Form App, using *Covidtracking.com* as a data source. For our sample application, we chose to make a call to the *CovidTracking.com* API for current state data. In order to translate the JSON response from this API, we developed a model C# class called **StateDate.** (See figure below)

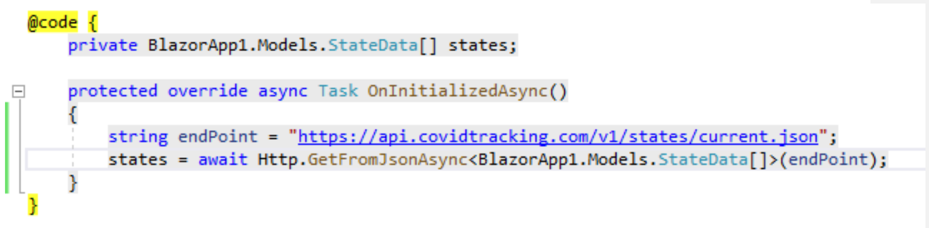


```
{
    2 references
    public class StateData
    {
        0 references | 0 exceptions
        public int date { get; set; }
        1 reference | 0 exceptions
        public string state { get; set; }
        1 reference | 0 exceptions
        public int? positive { get; set; } //confirmed + probable COVID cases
        1 reference | 0 exceptions
        public int? negative { get; set; } //negative COVID tests
        1 reference | 0 exceptions
        public int? positiveIncrease { get; set; } //new cases (confirmed + probable)
        1 reference | 0 exceptions
        public int? hospitalizedCurrently { get; set; } //currently hospitalized
        1 reference | 0 exceptions
        public int? inIcuCurrently { get; set; } //currently in icu with COVID
        1 reference | 0 exceptions
        public int? death { get; set; } //confirmed + probable
        1 reference | 0 exceptions
        public int? deathConfirmed { get; set; } //new deaths
        1 reference | 0 exceptions
        public int? deathProbable { get; set; } //death probable
        1 reference | 0 exceptions
        public int? deathIncrease { get; set; } //new deaths
        0 references | 0 exceptions
        public string dataQualityGrade { get; set; }
        0 references | 0 exceptions
        public string fips { get; set; } //state fips code
        0 references | 0 exceptions
        public string lastUpdateEt { get; set; }
    }
}
```

Fig. 2. Object model representation of JSON object

4.3 De-serialization of JSON API Response

De-serialization is the process of transforming a JSON API response for example into a useable programming object, with properties or variables. Using our proof of concept as an example, we de-serialized COVID-19 data from a JSON RESTful API response into the **StateData** object displayed above. In order to load data from the *CovidTracking.com* data API we added some code to one of our .razor pages, which consist of a combination of C#, HTML, CSS and JavaScript. On the razor page, we implemented the following code to de-serialize the JSON response into an array of **StateData** objects:



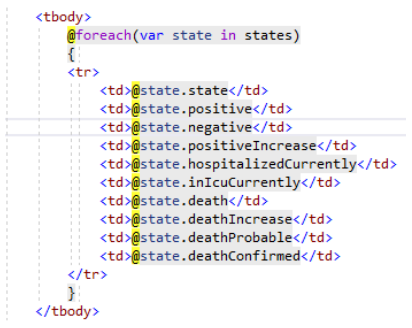
```
@code {
    private BlazorApp1.Models.StateData[] states;

    protected override async Task OnInitializedAsync()
    {
        string endPoint = "https://api.covidtracking.com/v1/states/current.json";
        states = await Http.GetFromJsonAsync<BlazorApp1.Models.StateData[]>(endPoint);
    }
}
```

Fig. 6. Deserialization of JSON API response into API Data Model

4.4 Making a RESTful API call

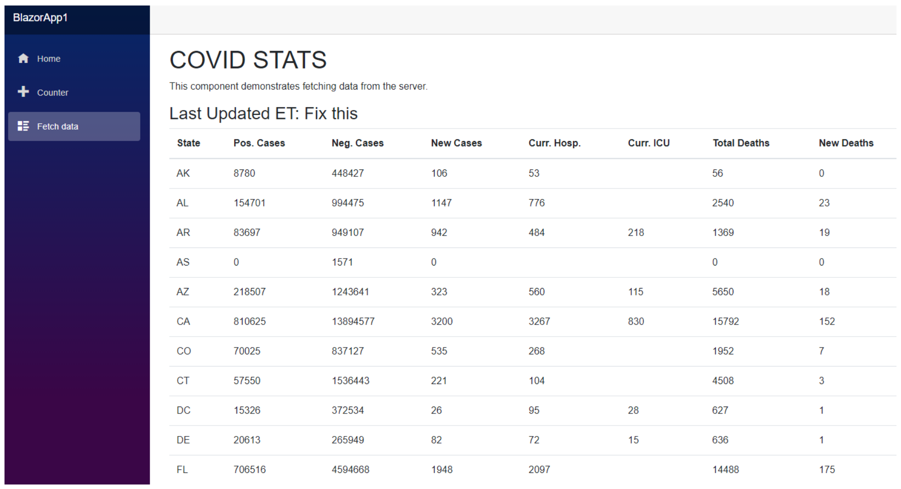
In the code snippet above, we are loading in a JSON string from a public end point and de-serializing this data into an array of **StateData** objects which are then display on-screen. We should mention that we referenced the following .NET libraries to execute data de-serialization and make API calls: System.Net.Http and System.Net.Http.Json. In order to display the data returned by the API, we populated HTML table rows with the state data and filled out the appropriate table header names. (See figure below)



```
<tbody>
    @foreach(var state in states)
    {
    <tr>
        <td>@state.state</td>
        <td>@state.positive</td>
        <td>@state.negative</td>
        <td>@state.positiveIncrease</td>
        <td>@state.hospitalizedCurrently</td>
        <td>@state.inIcuCurrently</td>
        <td>@state.death</td>
        <td>@state.deathIncrease</td>
        <td>@state.deathProbable</td>
        <td>@state.deathConfirmed</td>
    </tr>
    }
</tbody>
```

Fig. 7. Dynamically populating HTML table in .Razor page

This proof of concept for our Capstone Proposal resulted in our first working API call and Blazor web application prototype, which displayed US current state COVID-19 statistics. (See figure below)



COVID STATS

This component demonstrates fetching data from the server.

Last Updated ET: Fix this

State	Pos. Cases	Neg. Cases	New Cases	Curr. Hosp.	Curr. ICU	Total Deaths	New Deaths
AK	8780	448427	106	53		56	0
AL	154701	994475	1147	776		2540	23
AR	83697	949107	942	484	218	1369	19
AS	0	1571	0			0	0
AZ	218507	1243641	323	560	115	5650	18
CA	810625	13894577	3200	3267	830	15792	152
CO	70025	837127	535	268		1952	7
CT	57550	1536443	221	104		4508	3
DC	15326	372534	26	95	28	627	1
DE	20613	265949	82	72	15	636	1
FL	706516	4594668	1948	2097		14488	175

Fig. 3. *Proof of concept application, demonstrates call to REST API endpoint and displays data*

4.5 Creating an Appsettings.json file for Blazor Apps

4.6 Adding a Service that makes RESTful API calls

4.7 Injecting RESTful API service into .Razor pages

4.8 Form Entry within .Razor pages

4.9 CSS/Bootstrap for Blazor Applications

5. Conclusion/Summary/Future Work

- Finish implementation section
- Finish conclusion section after finalizing project code
- Add Cover Page and Abstract

- Finish Index page
    - Add project information such as Github repo and paper abstract
- Add additional Bootstrap CSS styling to our Blazor pages.
- Refactor code as necessary

5.1 TBD

6. References

Ting, D.S.W., Carin, L., Dzau, V. et al. Digital technology and COVID-19. Nat Med 26, 459–461 (2020). https://doi.org/10.1038/s41591-020-0824-5

A. Agocs and J. L. Goff, "A web service based on RESTful API and JSON Schema/JSON Meta Schema to construct knowledge graphs," 2018 International Conference on Computer, Information and Telecommunication Systems (CITS), Colmar, 2018, pp. 1-5, doi: 10.1109/CITS.2018.8440193.

Abeler, Johannes et al. "COVID-19 Contact Tracing and Data Protection Can Go Together." *JMIR mHealth and uHealth* vol. 8,4 e19359. 20 Apr. 2020, doi:10.2196/19359

E. Gurra, "Designing a Framework with Test-Driven Development: A Journey", in IEEE Software, vol. 31, no. 1, pp.9-14, Jan.-Feb. 2014, doi:10.1109/M.S2014.3 .

Neumann, Andy & Laranjeiro, Nuno & Bernardino, Jorge. (2018). An Analysis of Public REST Web Service APIs. IEEE Transactions on Services Computing. PP. 1-1. 10.1109/TSC.2018.2847344.

Agocs, Adam & Legoff, Jean-Marie. (2018). A web service based on RESTful API and JSON Schema/JSON Meta Schema to construct knowledge graphs.

Renner, Aaron. (2020). Study of Network Traffic from Java Chat Application. 10.13140/RG.2.2.29547.69923/1.

Edited by Robinson Meyer and Alexis Madrigal, *The COVID Tracking Project*, Mar. 2020, covidtracking.com/

"Blazor Tutorial: Build Your First App." *Microsoft*, 2020, dotnet.microsoft.com/learn/aspnet/blazor-tutorial/install.

"The Collaboration Platform for API Development." *Postman*, Postman Inc., 2020, www.postman.com