

Caraga Region COVID-19 Mapping: A Web Application for Interactive Map of COVID-19 cases in CARAGA Region

IT 109 SYSTEMS INTEGRATION AND ARCHITECHTURE 1

**Castro, Cedric,
Castro, Ryan Joel,
Fernando, Alvin,
Pagaran, John Bryan
Programmers**

I. Project Introduction

Caraga Region COVID 19 Mapping is a web application that displays the current cases of COVID 19 including deaths and recoveries. It has 2 way of displaying which is by city and overall cases in Caraga region. When the user wants to display cases of a certain city, example "Butuan" the user must put the city name into the search box then if the system finds a match from the server then it will return the geographical bounds that visible in the current map.

Caraga Region COVID 19 Mapping is written using jQuery and PHP, and uses leaflet that serves as API for the interactive map. The application also used XAMPP to host local web server. Geo-server must be run in order for the application to generate the geospatial data. It has a minimalist design, showing all the functionalities in the right part of the web application and a big map in the middle which the user will use to navigate and it has features that display all the cases including deaths and recoveries of covid-19 in Caraga region.

II. Feasibility Investigation

II.1 Database

The web application is using PostgreSQL for the database. The data that will be retrieve from the database will be displayed on the interface of the application. It will sum all the active cases, recoveries, and deaths of the entire Caraga Region.

Figure 1

```
$host = "localhost";      //host
$port = "5433";          //port
$user = "postgres";      //username
$pass = "1234";          //password
$db = "covid19";         //database name

/**Creating connection to PostgreSQL */
$con = pg_connect("host=$host port=$port dbname=$db user=$user password=$pass")
      or die ("Could not connect to server\n");
```

Figure 1: Shows how to open and create connection with PostgreSQL.

Figure 2

```
//query to get the total active cases, recoveries and deaths
$query = "SELECT SUM(active_cases),SUM(recoveries),SUM(deaths) FROM caragaregion_municipalities;";
$rs = pg_query($con, $query) or die("Cannot execute query: $query\n");

//if there is return, values will be saved to the variables
while ($row = pg_fetch_row($rs)) {
    $active_cases = $row[0];
    $recoveries = $row[1];
    $deaths = $row[2];
}
```

Figure 2: Shows the query to retrieve the total active cases, recoveries, and deaths.

After retrieving the total active cases, recoveries, and deaths, the data will be displayed on the web application with javascript counter animation.

Figure 3

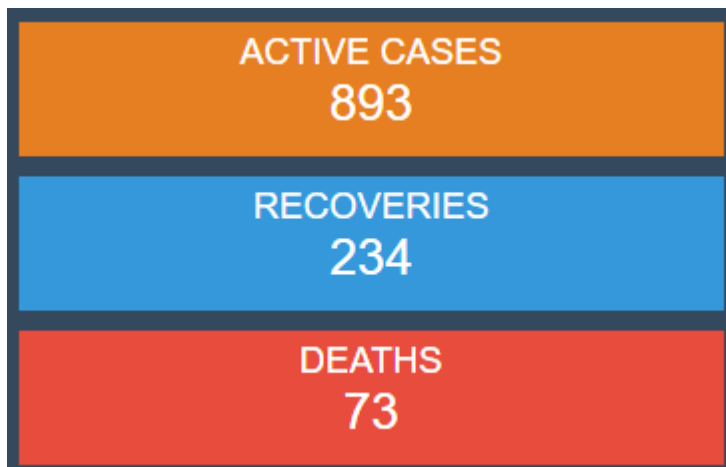


Figure 3: Shows the total active cases, recoveries, and deaths displayed on the web application

Figure 4

```
//Total records
$('.active_value').text(<?php echo $active_cases; ?>);
$('.rec_value').text(<?php echo $recoveries; ?>);
$('.death_value').text(<?php echo $deaths; ?>);

//Counter animation
$('.active_value, .rec_value, .death_value').each(function () {
    var $this = $(this);
    jQuery({ Counter: 0 }).animate({ Counter: $this.text() }, {
        duration: 4500,
        easing: 'swing',
        step: function () {
            $this.text(Math.ceil(this.Counter));
        }
    });
});
```

Figure 4: Shows the algorithm on to how display the active cases, recoveries, and deaths on the web interface.

For safety purposes, we have to close the connection of the database to prevent any other objects reusing the same connection.

Figure 5

```
//closing the connection
pg_close($con);
```

Figure 5: Shows how to close the connection from the database.

II.2 Leaflet

Caraga Region COVID 19 Mapping is using an interactive map API called leaflet (<https://www.leafletjs.com>) to display the map in the web application.

Figure 6

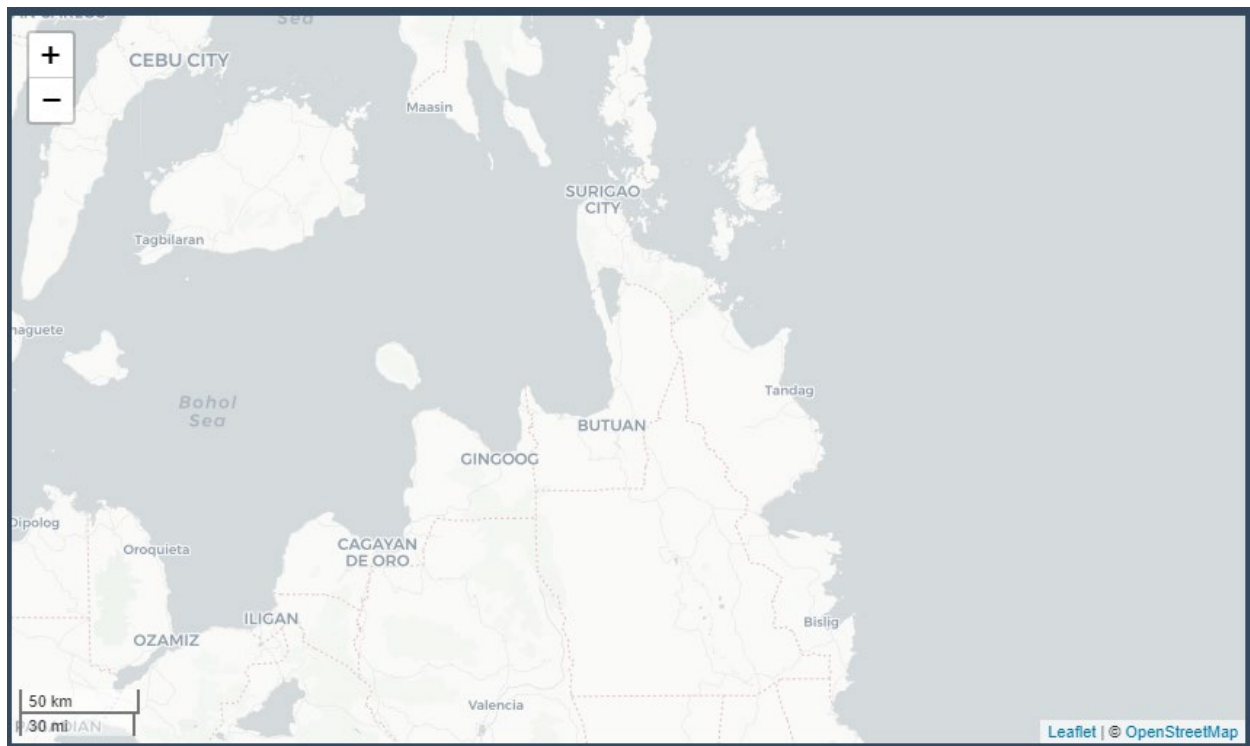


Figure 6: Shows the generated map on the web application using leaflet.

Before you can use the leaflet, you have to include first the leaflet library. You can save the library from www.leafletjs.com and use it offline.

Figure 7

```
<script type="text/javascript" src="leaflet/leaflet.js"></script>
```

Figure 7: Shows how we include the library leaflet in our web applications.

Figure 8

```
var map = L.map('map').setView([9.1204, 125.59], 8);  
var basemap = L.tileLayer('https://cartodb-basemaps-  
{s}.global.ssl.fastly.net/light_all/{z}/{x}/{y}.png', {  
  attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">Ope  
nStreetMap </a> contributors'  
}).addTo(map);
```

Figure 8: Shows how the map is displayed into the web application using the library leaflet.

II.3 GeoServer

The web application is connected to GeoServer which generates a map layer and a map legend. The generated layer will be overlaid to the leaflet map.

Figure 8

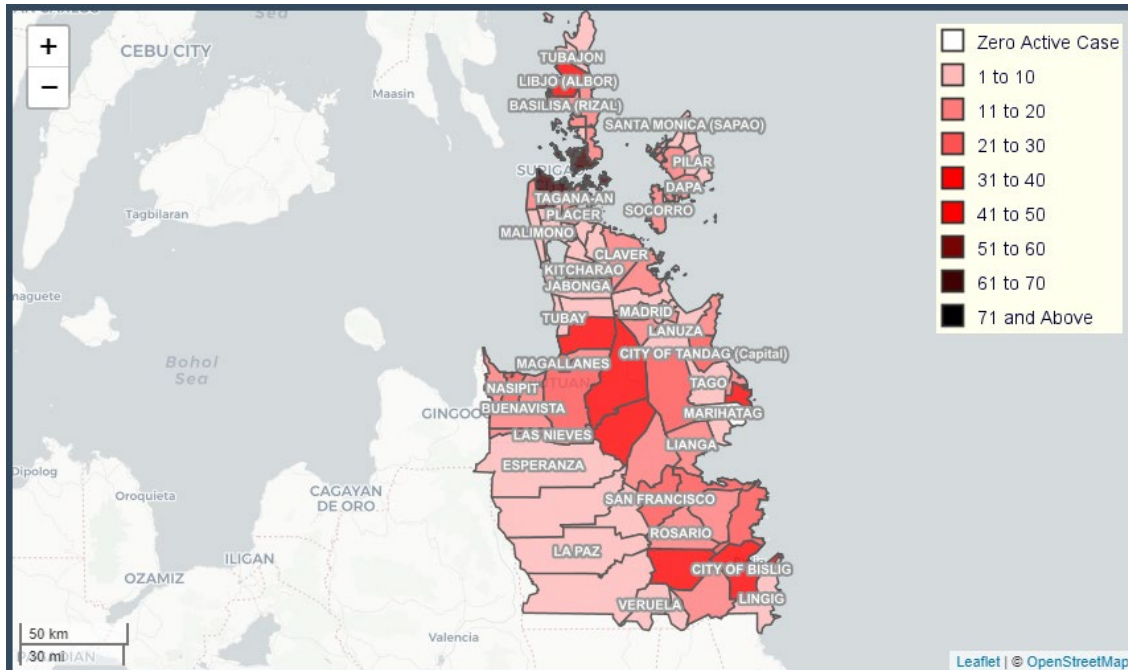


Figure 8: Shows the map from the leaflet overlaid by a layer and a legend from the GeoServer.

Figure 9

```
legend.onAdd = function (map) {  
    var div = L.DomUtil.create('div', 'info legend');  
    div.innerHTML = '';  
    return div;  
};  
legend.addTo(map);
```

Figure 9: Shows how the legend from the GeoServer is added to the map.

Figure 10

```
var owsURI = 'http://127.0.0.1:8080/geoserver/covid19/ows';
var municipalities;
function load_municipalities(){
    municipalities = L.tileLayer.wms(owsURI, {
        layers: Selected_layer,
        format: 'image/png',
        transparent: true,
        attribution: "",
        zIndex: 100,
        opacity: .8,
        version: '1.1.0',
    }).addTo(map);
}

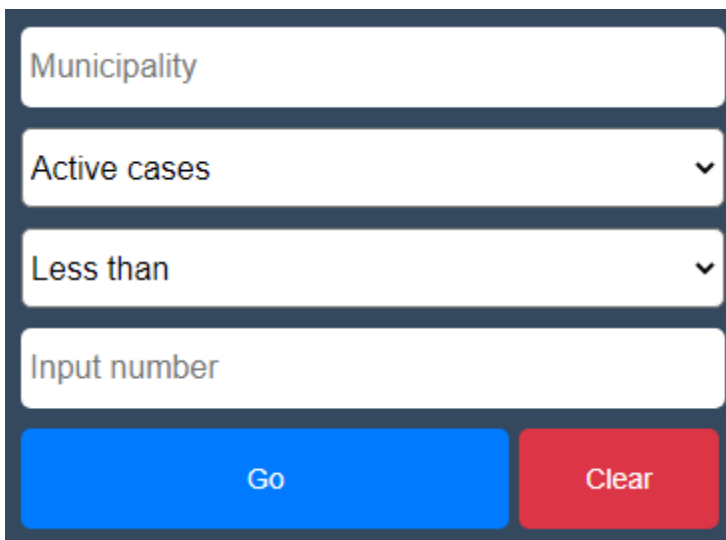
load_municipalities();
```

Figure 10: Shows how the layer from the GeoServer is added to the map.

II.4 Common Query Language (CQL)

The next algorithm will filter the map using CLQ filtering and will send a WFS request to the GeoServer. It will display the parameters requested by the user. CQL can filter the generated layer by municipality name, active cases, recoveries, deaths and greater or lesser value of the inputted value of the user.

Figure 11



The image shows a web application interface for the Common Query Language (CQL) filter. It consists of four input fields stacked vertically, each with a light blue border and a light gray placeholder text. The first field is labeled 'Municipality'. The second field is labeled 'Active cases' and has a downward arrow icon on its right side. The third field is labeled 'Less than' and also has a downward arrow icon on its right side. The fourth field is labeled 'Input number'. Below these fields are two buttons: a blue button labeled 'Go' and a red button labeled 'Clear'.

Figure 11: Shows the CQL filter of the web application.

Figure 12

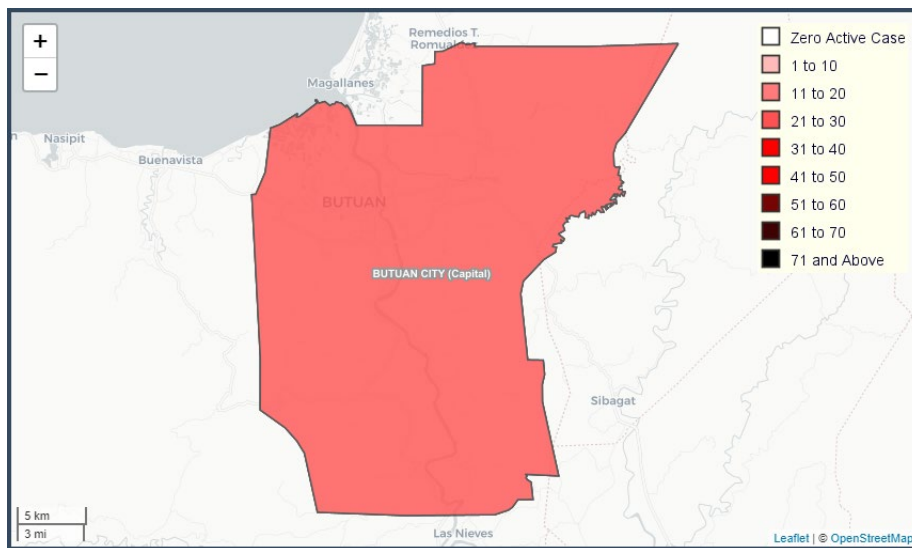


Figure 12: Shows filtered layer by municipality name Butuan City.

Figure 13

```
//Setting CQL filter from the input values
function set_cql(textInput,greaterLess, numberInput, typeOfQuery) {
  var cql_filter = [];
  if (textInput){
    cql_filter.push("mun_name LIKE '%" +textInput+"%'")
  }
  if (numberInput){
    cql_filter.push(typeOfQuery+" "+greaterLess+" "+numberInput)
  }
  if (cql_filter.length == 2){
    cql_filter.splice(1, 0, "AND");
  }
  if (cql_filter.length==0){
    cql_filter.push("1=1")
  }
  municipalities.setParams({
    CQL_FILTER: cql_filter.join(' ')
  })
  wfsRequest(cql_filter.join(' '));
}
```

Figure 8: Shows how to set the CQL filter from the queries or parameters inserted by the user.


```

function wfsRequest(filter){
    function getJsonCurrUtil(data){}
    var defaultParameters = {
        service: 'WFS',
        version: '2.0.0',
        request: 'GetFeature',
        typeName: Selected_layer,
        maxFeatures: 500,
        outputFormat: 'application/json',
        cql_filter: filter,
        format_options: 'callback: getJson',
        srsName: 'EPSG:4326'
    }
    var parameters = L.Util.extend(defaultParameters);
    var URL = owsURI + L.Util.getParamString(parameters);
    $.ajax({
        url: URL,
        dataType: 'json',
        jsonpCallback: 'getJson',
        beforeSend : function (){
            $(".mapOverlay").addClass("active");
            $('#maploader').show();
        },
        success: function (response) {
            $('#maploader').hide();
            $(".mapOverlay").removeClass("active");
            if(response.features.length > 0){
                WFSLayer = L.geoJson(response, {
                    style: function (feature) {
                        return {
                            stroke: false,
                            fillColor: null,
                            fillOpacity: 0
                        };
                    }
                }).addTo(map);
                map.fitBounds(WFSLayer.getBounds());
            }else{
                $("#mun_message").html($('#textInput').val()
                + " not found in CARAGA REGION");
                $(".cover").addClass("active");
                $(".popup-overlay, .popup-content").addClass("active");
                load_municipalities();
                map.setView([9.1204, 125.59], 8);
            }
        }
    })
}

```

Figure 13: Shows the algorithm of the CQL filter.

II. 5 Map onClick feature

The web application can display the active cases, recoveries, and deaths of a municipality when clicked. A popup will appear with the information of that municipality.

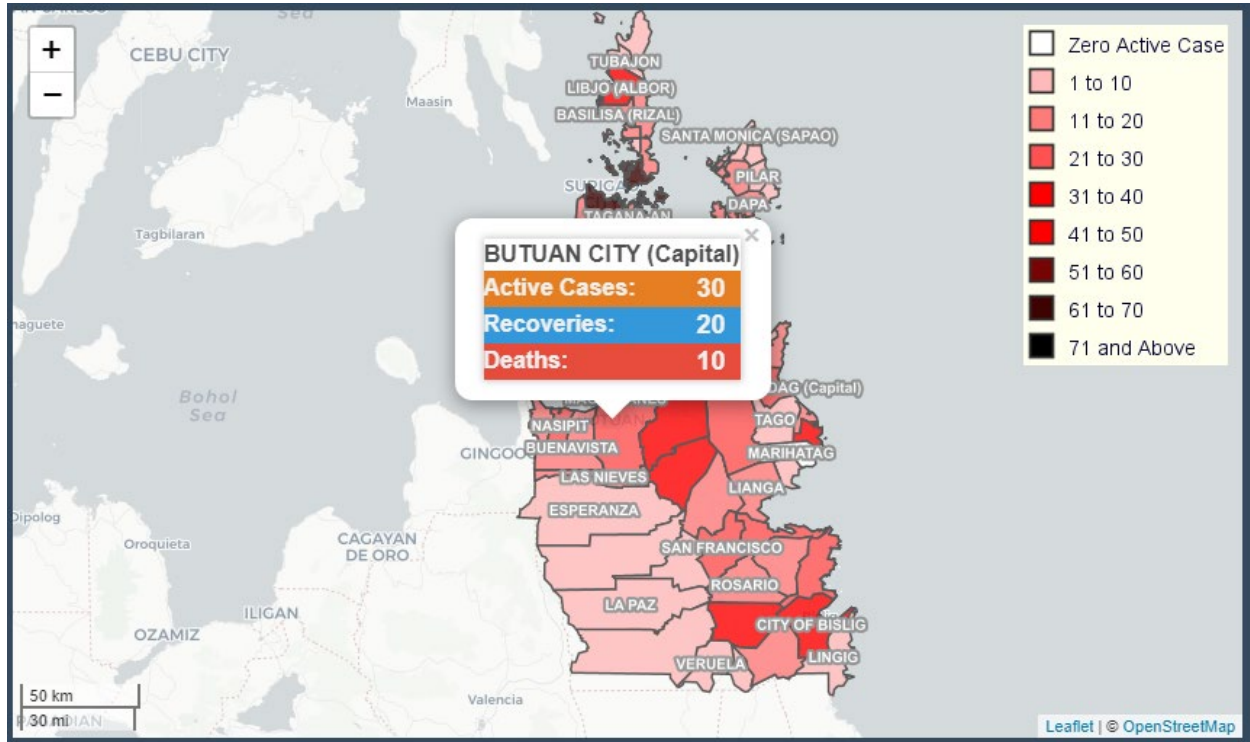


Figure 14: Shows a popup with municipality name, active cases, recoveries, and deaths of the clicked municipality.

II.6 Table view of the entire Caraga Region

The web application can display the entire information of covid-19 in the entire Caraga region. It will be displayed on a table with municipality name, current active cases, recoveries, and deaths.

Figure 15

Show entries				Search
10				
Municipality / City	Active Cases	Recoveries	Deaths	
ALEGRIA	10	3	2	
BACUAG	7	3	1	
BAROBO	24	14	1	
BASILISA (RIZAL)	15	8	3	
BAYABAS	13	3	0	
BUENAVISTA	15	12	1	
BUNAWAN	34	17	4	
BURGOS	5	2	0	
BUTUAN CITY (Capital)	30	20	10	
CAGDIANAO	17	5	1	
Showing 1 to 10 of 71 entries				Previous 1 2 3 4 5 ... 8 Next

Figure 15: Shows a table of active cases, recoveries, and deaths of covid-19 in the entire Caraga region.

Figure 16

```
$query = "SELECT mun_name, active_cases, recoveries, deathsFROM caragaregion_mu
nicipalities;";
$result = pg_query($con, $query) or die("Cannot execute query: $query\n");

while ($row = pg_fetch_row($result)) {
echo "<tr>";
echo "<td>" . $row [0]. "</td>";
echo "<td class = 'center'>" . $row [1]. "</td>";
echo "<td class = 'center'>" . $row [2]. "</td>";
echo "<td class = 'center'>" . $row [3]. "</td>";
echo "</tr>";

}

//closing the connection
pg_close($con);
```

Figure 16: Shows query how the data was retrieved from the database and display into the interface.

III. Requirements Specifications

III.1 Software Requirements

The following software is needed for the development of this application:

- Microsoft Visual Studio code to provide text editor.
- Web browser to display the application interface.
- Leaflet library for interactive maps.
- Geo-server as a server for the geospatial data.
- PostgreSQL for the database of the application
- XAMPP for the webserver.

III.2 Hardware Requirements

- One unit of personal computer or laptop.

III.3 Functional Requirements

- The application shall filter the map based on Municipality name and greater than or lesser than of the active cases, recoveries, deaths.
- The application shall display the total active cases, recoveries, and deaths for the entire Caraga region.
- The application shall display the municipality information such as active cases, recoveries, and deaths upon clicked.
- The application shall display the active cases, recoveries and deaths of the entire Caraga region in a table view or list view.

III.4 System Attributes

- Usability: The application is easy to use for new or infrequent user.
- Maintainability: Ease of adding or modifying code and data to the application.
- Reliability: The application is reliable enough because the developers get the data from reliable sources such as government and etc.
- Efficiency: The application is able to display the results in a short period of time.
- Testability: The application is easy to test.

GitHub Repository: