**UNIVERSITY OF TECHNOLOGY, MAURITIUS**

**SCHOOL OF INNOVATIVE TECHNOLOGIES AND ENGINEERING**

**DEPARTMENT OF INDUSTRIAL SYSTEM ENGINEERING**

**Module Name: Web Services**

**Module Code: WAT2124C**

**Cohort: BCNS/ 19A/ FT**

**Student Name: Arouven POOLIAN**

**Student ID: 1903\_16737**

Table of Contents

[Chapter 1: Introduction 1](#_Toc74250892)

[Chapter 2: Design and implementation 2](#_Toc74250893)

[2.1: Core functionalities 2](#_Toc74250894)

[2.1.2: Xml 3](#_Toc74250895)

[2.1.3: Json 4](#_Toc74250896)

[2.2: Additional functionalities 6](#_Toc74250897)

[2.2.1: Individual Map 6](#_Toc74250898)

[2.2.2: Get location 7](#_Toc74250899)

[2.2.3: regional map 7](#_Toc74250900)

[2.2.4: World map 8](#_Toc74250901)

[2.2.5: Highlight magnitude 11](#_Toc74250902)

[2.2.6: Js url builder 14](#_Toc74250903)

[2.2.7: Fill elements 18](#_Toc74250904)

[2.2.8: Details 20](#_Toc74250905)

[Chapter 3: Testing 21](#_Toc74250906)

[Chapter 4: Conclusion 22](#_Toc74250907)

**List of Figures**

[Figure 2.1: Data table. 2](#_Toc74250908)

[Figure 2.2: xml function. 3](#_Toc74250909)

[Figure 2.3: html record builder. 4](#_Toc74250910)

[Figure 2.4: json function. 5](#_Toc74250911)

[Figure 2.5: Data table outputted from xml and json. 5](#_Toc74250912)

[Figure 2.6: Worldwide map. 8](#_Toc74250913)

[Figure 2.7: Java script map. 10](#_Toc74250914)

[Figure 2.8: Java script markers variable outputted from php. 11](#_Toc74250915)

[Figure 2.9: Function to return css classes from magnitude values. 12](#_Toc74250916)

[Figure 2.10: html record builder with css class. 12](#_Toc74250917)

[Figure 2.11: css for magnitudes. 13](#_Toc74250918)

[Figure 2.12: Output of highlights and css. 13](#_Toc74250919)

[Figure 2.13: colors and magnitudes. 14](#_Toc74250920)

[Figure 2.14: Date time picker. 15](#_Toc74250921)

[Figure 2.15: legend of the date parameters. 15](#_Toc74250922)

[Figure 2.15: Digital clock. 15](#_Toc74250923)

[Figure 2.15: Digital clock js code. 16](#_Toc74250924)

[Figure 2.15: format selector. 16](#_Toc74250925)

[Figure 2.15: alert level selector. 17](#_Toc74250926)

[Figure 2.15: legend of the alert levels. 17](#_Toc74250927)

[Figure 2.15: js url builder. 18](#_Toc74250928)

[Figure 2.15: js html elements filler. 19](#_Toc74250929)

[Figure 2.15: Details button. 20](#_Toc74250930)

[Figure 2.15: redirect to detailed page. 20](#_Toc74250931)

# Introduction

# Design and implementation

## Core functionalities

The best way to display the 2 core functionalities is to make use of data table (shown in Figure 2.1) learnt in previous module internet programing ii. In this way, the user can do search, and sorting on any columns.

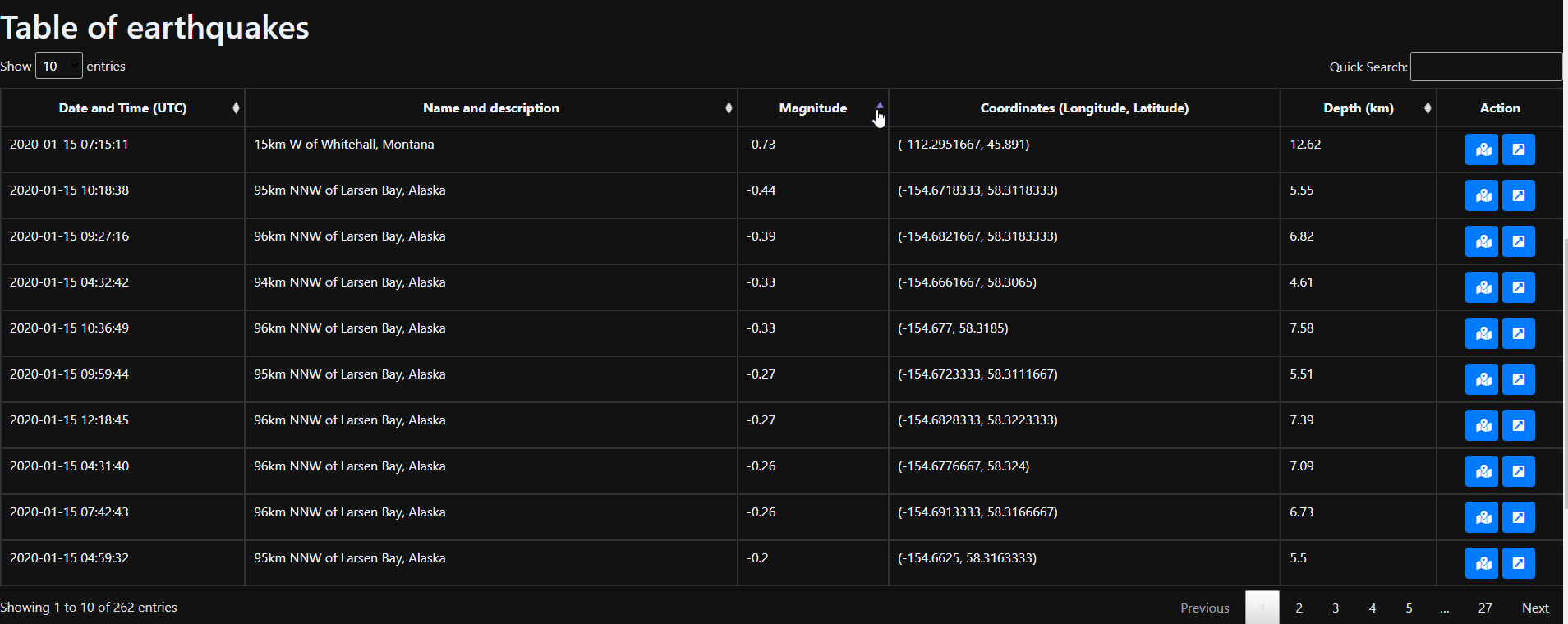


Figure 2.1: Data table.

The steps that will be used are as follows:

Get the file (xml or json) downloaded.

Extract wanted data (description, magnitude…) from the file.

Use the data to add table records through php.

### Xml

OOP approach will be used to reduce the amount of lines of codes. A function for xml url has been made as shown in the snippet below (Figure 2.2). This will gather the xml file from the url and save it in a variable $xml. With a “foreach” loop, the required variables will be retrieved and pass to a function “fillTable” which in its turn will build the html table records as shown in Figure 2.3.

$xml = @simplexml\_load\_file($this->url);

foreach ($xml->children()->children() as $event) {

    if (empty($event->description->text)) { //skip the one with empty data -- prevent wrong filling of datatable

        break;

    }

    $description = $event->description->text;

    $datetime =  strval(date("Y-m-d H:i:s", strtotime($event->origin->time->value))); // get the date in string and convert it to date

    $magnitude = $event->magnitude->mag->value;

    $longitude = $event->origin->longitude->value;

    $latitude = $event->origin->latitude->value;

    $depth = strval(((float)$event->origin->depth->value) / 1000); //convert to km

    $url\_components = parse\_url($event['publicID']); // convert the text into url

    parse\_str($url\_components['query'], $params); // store all queries in $params

    $url = 'https://earthquake.usgs.gov/earthquakes/eventpage/' . $params['eventid'];

    $this->fillTable($datetime, $description, $magnitude, $longitude, $latitude, $depth, $url);

}

Figure 2.2: xml function.

echo '

<tr>

    <td>' . $this->datetime . '</td>

    <td>' . $this->description . '</td>

    <td ' . $this->magnitude . '</td>

    <td>' . strval("($this->longitude, $this->latitude)") . '</td>

    <td>' . $this->depth . '</td>

    <td style="width: 100px;">

        <div style="text-align: center;">

            <button type="button"

            class="btn btn-primary"

            style="width: 40px;"

            data-toggle="modal"

            data-target="#myModal"

            data-lat=' . $this->latitude . '

            data-lng=' . $this->longitude . '

            title="open map ' . $this->description . '">

                <i class="fas fa-map-marked-alt"></i>

            </button>

            <button type="button"

            class="btn btn-primary"

            style="width: 40px;"

            title="Open details"

            onclick="location.href = \'' . $this->url . '\';">

                <i class="fa fa-external-link-square-alt"></i>

            </button>

        </div>

    </td>

</tr>

';

Figure 2.3: html record builder.

### Json

The use of oop has made this task easier. By just adding another function for the json file shown in Figure 2.4, the same output will be received as displayed in Figure 2.5.

$json = file\_get\_contents($this->url); //get the content in the json file

$data = json\_decode($json, true); //get the json out of it and true to be able to use the key in the foreach loop

foreach ($data['features'] as $key => $value) { //key will return the position in the array

    $datetime = date("Y-m-d H:i:s", substr($data['features'][$key]['properties']['time'], 0, 10)); // get the date in string and convert it to date

    $description = $data['features'][$key]['properties']['place'];

    $magnitude = $data['features'][$key]['properties']['mag'];

    $longitude = $data['features'][$key]['geometry']['coordinates'][0];

    $latitude = $data['features'][$key]['geometry']['coordinates'][1];

    $depth = $data['features'][$key]['geometry']['coordinates'][2];

    $url = $data['features'][$key]['properties']['url'];

    $this->fillTable($datetime, $description, $magnitude, $longitude, $latitude, $depth, $url);

}

Figure 2.4: json function.

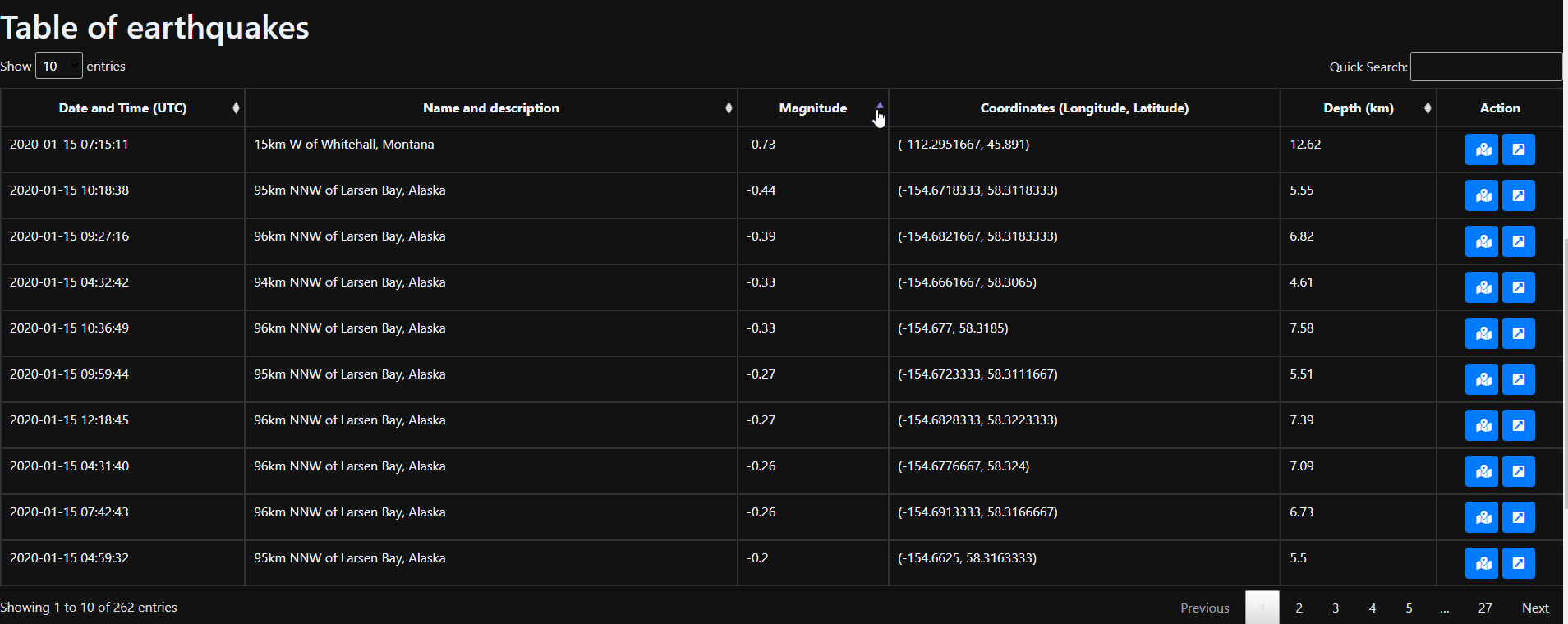
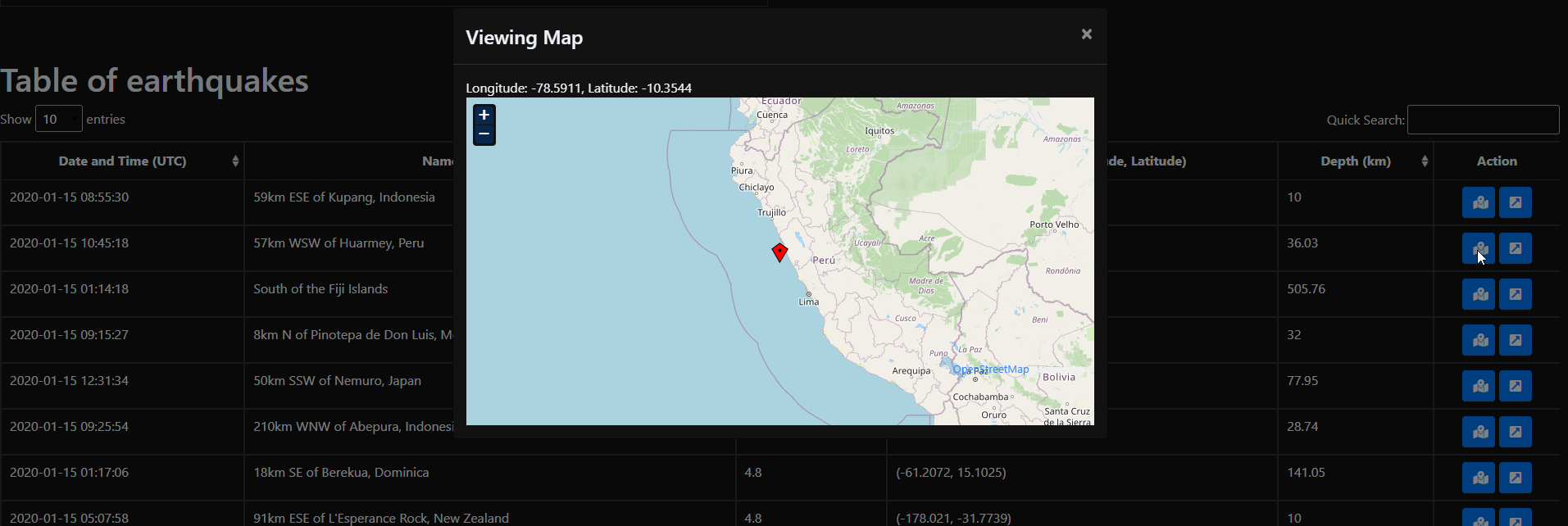


Figure 2.5: Data table outputted from xml and json.

## Additional functionalities

### Individual Map

On clicking on the map, there will be a modal with the map and the location





### Get location

### regional map

### World map

To accompany the whole data table a map is made with all the locations and markers as displayed in Figure 2.6. Figure 2.7 shows the approach to make the map. Js was used with openstreetmap. Why this map – because it is the most famous and well defined open source map available on the internet. There is even a wiki. The markers are received from php which has built a variable for the js as shown in Figure 2.8.

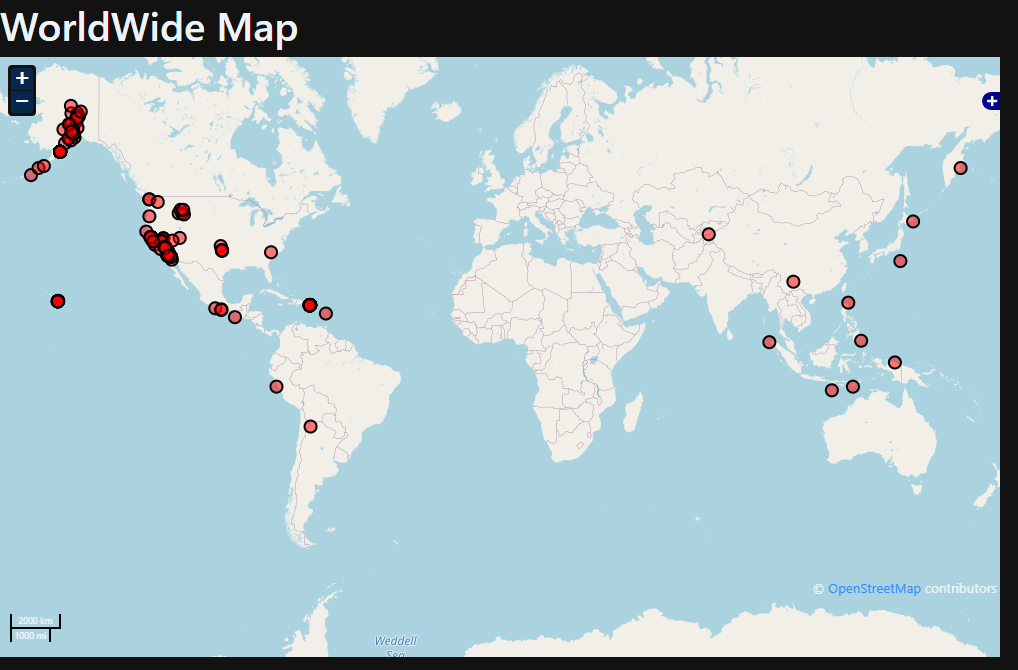


Figure 2.6: Worldwide map.

//start building big map

function detectQueryString() {

    var currentQueryString = window.location.search;

    if (currentQueryString) {

        return true;

    } else {

        return false;

    }

};

function bigMapBuilder() {

    epsg4326 = new OpenLayers.Projection("EPSG:4326")

    bmap = new OpenLayers.Map({

        div: "mapdiv",

        displayProjection: epsg4326 // With this setting, lat and lon are displayed correctly in MousePosition and permanent anchor

    });

    bmap.addLayer(new OpenLayers.Layer.OSM());

    bmap.addLayer(new OpenLayers.Layer.OSM("Wikimedia",

        ["https://maps.wikimedia.org/osm-intl/${z}/${x}/${y}.png"], {

        attribution: "&copy; <a href='http://www.openstreetmap.org/'>OpenStreetMap</a> and contributors, under an <a href='http://www.openstreetmap.org/copyright' title='ODbL'>open license</a>. <a href='https://www.mediawiki.org/wiki/Maps'>Wikimedia's new style (beta)</a>",

        "tileOptions": {

            "crossOriginKeyword": null

        }

    }));

    bmap.addControls([

        new OpenLayers.Control.MousePosition(),

        new OpenLayers.Control.ScaleLine(),

        new OpenLayers.Control.LayerSwitcher(),

        new OpenLayers.Control.Permalink({

            anchor: true

        })

    ]);

    projectTo = bmap.getProjectionObject(); //The map projection (Spherical Mercator)

    var lonLat = new OpenLayers.LonLat(8.0, 50.3).transform(epsg4326, projectTo);

    var zoom = 1;

    if (!bmap.getCenter()) {

        bmap.setCenter(lonLat, zoom);

    }

    var colorList = ["red"];

    var layerName = [markers[0][2]];

    var styleArray = [new OpenLayers.StyleMap({

        pointRadius: 6,

        fillColor: colorList[0],

        fillOpacity: 0.5

    })];

    var vectorLayer = [new OpenLayers.Layer.Vector(layerName[0], {

        styleMap: styleArray[0]

    })]; // First element defines first Layer

    var j = 0;

    for (var i = 1; i < markers.length; i++) {

        if (!layerName.includes(markers[i][2])) {

            j++;

            layerName.push(markers[i][2]); // If new layer name found it is created

            styleArray.push(new OpenLayers.StyleMap({

                pointRadius: 6,

                fillColor: colorList[j % colorList.length],

                fillOpacity: 0.5

            }));

            vectorLayer.push(new OpenLayers.Layer.Vector(layerName[j], {

                styleMap: styleArray[j]

            }));

        }

    }

    //Loop through the markers array

    for (var i = 0; i < markers.length; i++) {

        var lon = markers[i][0];

        var lat = markers[i][1];

        var feature = new OpenLayers.Feature.Vector(

            new OpenLayers.Geometry.Point(lon, lat).transform(epsg4326, projectTo), {

            description: "marker number " + i

        });

        vectorLayer[layerName.indexOf(markers[i][2])].addFeatures(feature);

    }

    for (var i = 0; i < layerName.length; i++) {

        bmap.addLayer(vectorLayer[i]);

    }

};

$(document).ready(function () {//build the big map only if there is a query in the url

    if (!detectQueryString()) {

        xmlCore();

    }

    bigMapBuilder();

});

//end building big map

Figure 2.7: Java script map.

$this->setMarkers($longitude, $latitude); //building js var

//echo the javascript markers variables

function getjsMarkers()

{

    echo "<script>var markers=[$this->markers];</script>";

}

//filling the variable markers

function setMarkers($longitude, $latitude)

{

    $this->markers = $this->markers . "[$longitude,$latitude],";

}

Figure 2.8: Java script markers variable outputted from php.

### Highlight magnitude

To make the user more at ease with the gravity of the earthquake, colors are added to the magnitudes. This is done using a function (Figure 2.9) that will return a css class and then add the class to the respective magnitude as illustrated in Figure 2.10. the css file is now updated as shown in Figure 2.11.

//get the classes -- to help css

function earthquakeClass(string $value)

{

    // Class    Magnitude

    // Great    8 or more

    // Major    7 - 7.9

    // Strong   6 - 6.9

    // Moderate 5 - 5.9

    // Light    4 - 4.9

    // Minor    3 -3.9

    if ((int)$value >= 8) {

        return 'Great';

    } elseif ((int)$value == 7) {

        return 'Major';

    } elseif ((int)$value == 6) {

        return 'Strong';

    } elseif ((int)$value == 5) {

        return 'Moderate';

    } elseif ((int)$value == 4) {

        return 'Light';

    } elseif ((int)$value == 3) {

        return 'Minor';

    } else {

        return 'Negligible';

    }

}

Figure 2.9: Function to return css classes from magnitude values.

$highlightMagnitudeClasses = $this->earthquakeClass($this->magnitude);

echo '

<tr>

    <td>' . $this->datetime . '</td>

    <td>' . $this->description . '</td>

    <td class="' . $highlightMagnitudeClasses . '">' . $this->magnitude . '</td>

    <td>' . strval("($this->longitude, $this->latitude)") . '</td>

    <td>' . $this->depth . '</td>

    <td style="width: 100px;">

        <div style="text-align: center;">

            <button type="button"

            class="btn btn-primary"

            style="width: 40px;"

            data-toggle="modal"

            data-target="#myModal"

            data-lat=' . $this->latitude . '

            data-lng=' . $this->longitude . '

            title="open map ' . $this->description . '">

                <i class="fas fa-map-marked-alt"></i>

            </button>

            <button type="button"

            class="btn btn-primary"

            style="width: 40px;"

            title="Open details"

            onclick="location.href = \'' . $this->url . '\';">

                <i class="fa fa-external-link-square-alt"></i>

            </button>

        </div>

    </td>

</tr>

';

Figure 2.10: html record builder with css class.

/\* css for tables \*/

.Moderate,

.Strong {

  color: orange;

}

.Moderate,

.Strong,

.Major,.BoldMe,

.Great {

  font-weight: bold;

}

.Major,

.Great {

  color: red;

}

th {text-align: center;}

Figure 2.11: css for magnitudes.

The code added has outputted the desired outcome as illustrated in the magnitude column in Figure 2.12.

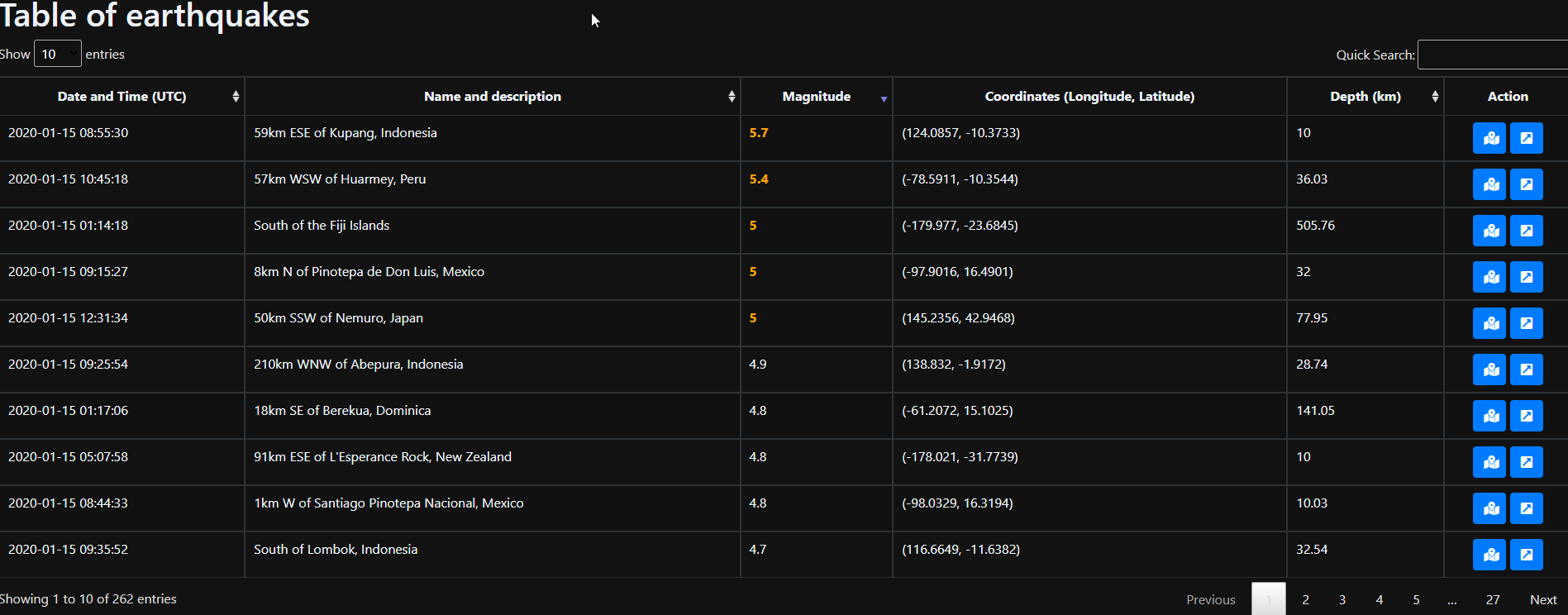


Figure 2.12: Output of highlights and css.

#### Legend of the highlighted magnitudes

A legend of the respective colors highlighted will help the user to make his/her choice while searching for earthquakes.

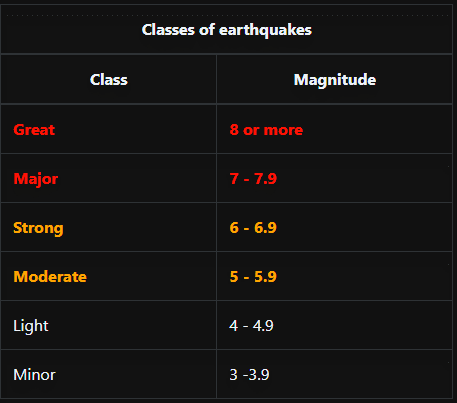


Figure 2.13: colors and magnitudes.

### Js url builder

Java script and jquery were used to build the url as per the user selected search. There are date time filter, format selector, and alert level selector.

#### Date and time filter

A date and time filter will help the user to pick up the wanted date rather than to insert it to the url. Both start and end dates are required to give the user the best experience as shown in Figure 2.14.

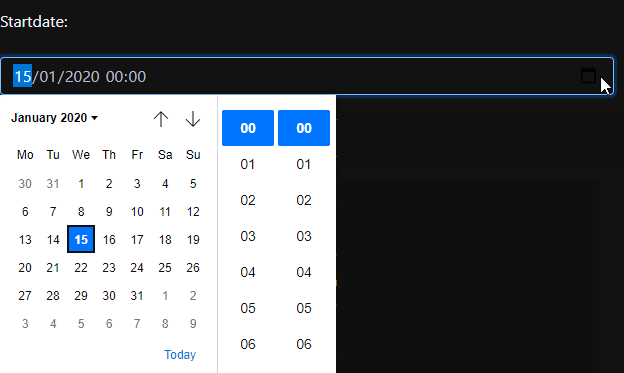


Figure 2.14: Date time picker.

#### Legend of the date parameters

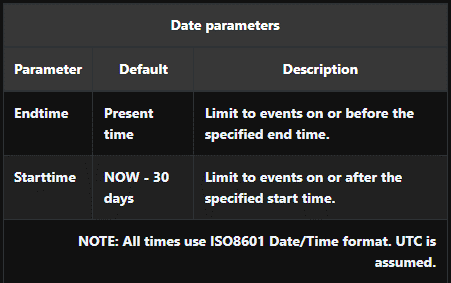


Figure 2.15: legend of the date parameters.

#### Digital UTC clock

As the time in the queried file (xml or json) uses UTC time, the user will have to edit his/her computer time. To avoid this, a clock is put on the website itself.



Figure 2.16: Digital clock.

//start clock

function currentTime() {

    var date = new Date(); // creating object of Date class

    var hour = date.getUTCHours();

    var min = date.getUTCMinutes();

    var sec = date.getUTCSeconds();

    hour = updateTime(hour);

    min = updateTime(min);

    sec = updateTime(sec);

    document.getElementById("clock").innerText = "UTC " + hour + " : " + min + " : " + sec; // adding time to the div

    var t = setTimeout(function () {

        currentTime()

    }, 1000); // setting timer

}

function updateTime(k) {

    if (k < 10) {

        return "0" + k;

    } else {

        return k;

    }

}

currentTime();

//end clock

Figure 2.17: Digital clock js code.

#### Format selector

This will make use of different format to query the server and get response in the respective format

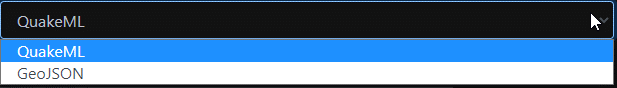


Figure 2.18: format selector.

#### Alert level selector

On selecting the alert it will display the power of the earthquakes

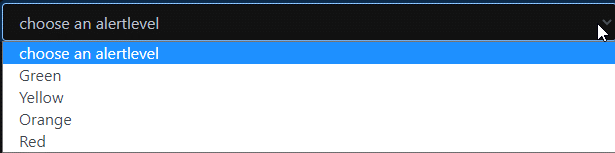


Figure 2.19: alert level selector.

#### Legend of the alert levels

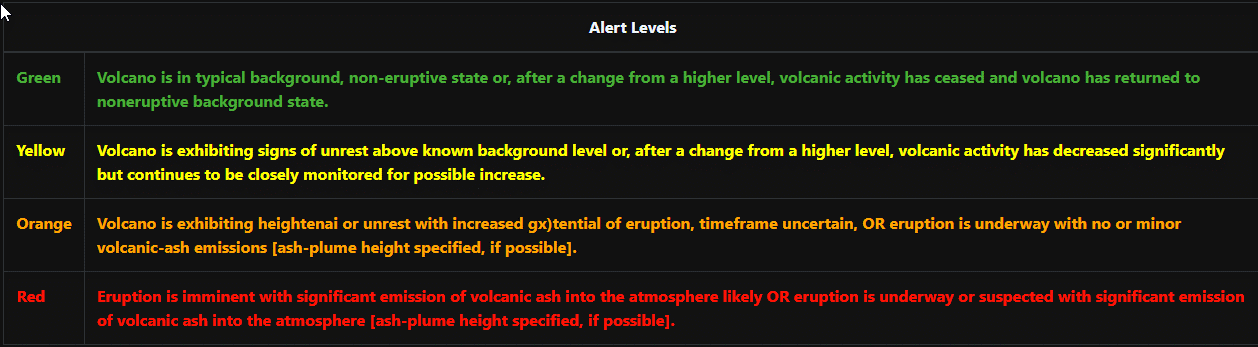


Figure 2.20: legend of the alert levels.

Once the search button is clicked, js will retrieving all the user information, and build the url then redirect to it.

function buildURL() {

    var output = window.location.pathname + "?url='https://earthquake.usgs.gov/fdsnws/event/1/query?";//build the initial url with the file.php followed by the query

    // get appropriate values from elements

    var format = $('#format').val();

    var starttime = $("#start\_date").val();

    var endtime = $("#end\_date").val();

    var longitude = $("#lon").val();

    var latitude = $("#lat").val();

    var maxradiuskm = $("#rad").val();

    var alertlevel = $('#alertlevel').val();

    var currenttime = new Date();

    //validation

    if ((Date.parse(starttime) > Date.parse(endtime)) || ((Date.parse(endtime) > Date.parse(currenttime)))) {//validation date

        alert("Invalid Date Range");

    } else if ((parseFloat(maxradiuskm) > 20001.6)) {// validation on the max radius

        alert("Invalid radius must be less than 20001.6");

    } else {// build the url

        output += "format=" + format;

        if (starttime) {

            output += "&starttime=" + starttime;

        }

        if (endtime) {

            output += "&endtime=" + endtime;

        }

        if (alertlevel) {

            output += "&alertlevel=" + alertlevel;

        }

        if (longitude) {

            output += "&longitude=" + longitude;

        }

        if (latitude) {

            output += "&latitude=" + latitude;

        }

        if ((maxradiuskm) && (longitude) && (latitude)) {

            output += "&maxradiuskm=" + maxradiuskm;

        }

        output += "'";//closing the url

        window.open(output, "\_self");//open the url in the current tab

        //alert(output);

    }

}

Figure 2.21: js url builder.

### Fill elements

The use of js was use to fill each elements in the advance search

//start fill html elements

function fillElements() {

    var url = window.location.href;// get current url

    url = url.substring(url.indexOf('%27') + 0);//reject text before %27

    url = url.replaceAll('%27', '');//remove all %27

    url = new URL(url);//change text to url

    // get appropriate queries from url

    var format = url.searchParams.get("format");

    var starttime = url.searchParams.get("starttime");

    var endtime = url.searchParams.get("endtime");

    var alertlevel = url.searchParams.get("alertlevel");

    var longitude = url.searchParams.get("longitude");

    var latitude = url.searchParams.get("latitude");

    var maxradiuskm = url.searchParams.get("maxradiuskm");

    //set the elements with the appropriate values from the query

    $('#format').val(format);

    if (starttime) {

        $("#start\_date").attr('value', starttime);

    }

    if (endtime) {

        $("#end\_date").attr('value', endtime);

    }

    if (alertlevel) {

        $('#alertlevel').val(alertlevel);

    }

    if (longitude) {

        $("#lon").val(longitude);

    }

    if (latitude) {

        $("#lat").val(latitude);

    }

    if (maxradiuskm) {

        $("#rad").val(maxradiuskm);

    }

}

//end filling html elements

Figure 2.22: js html elements filler.

### Details

Once detail is clicked, other web page will open with all the details of that earth quake.



Figure 2.23: Details button.

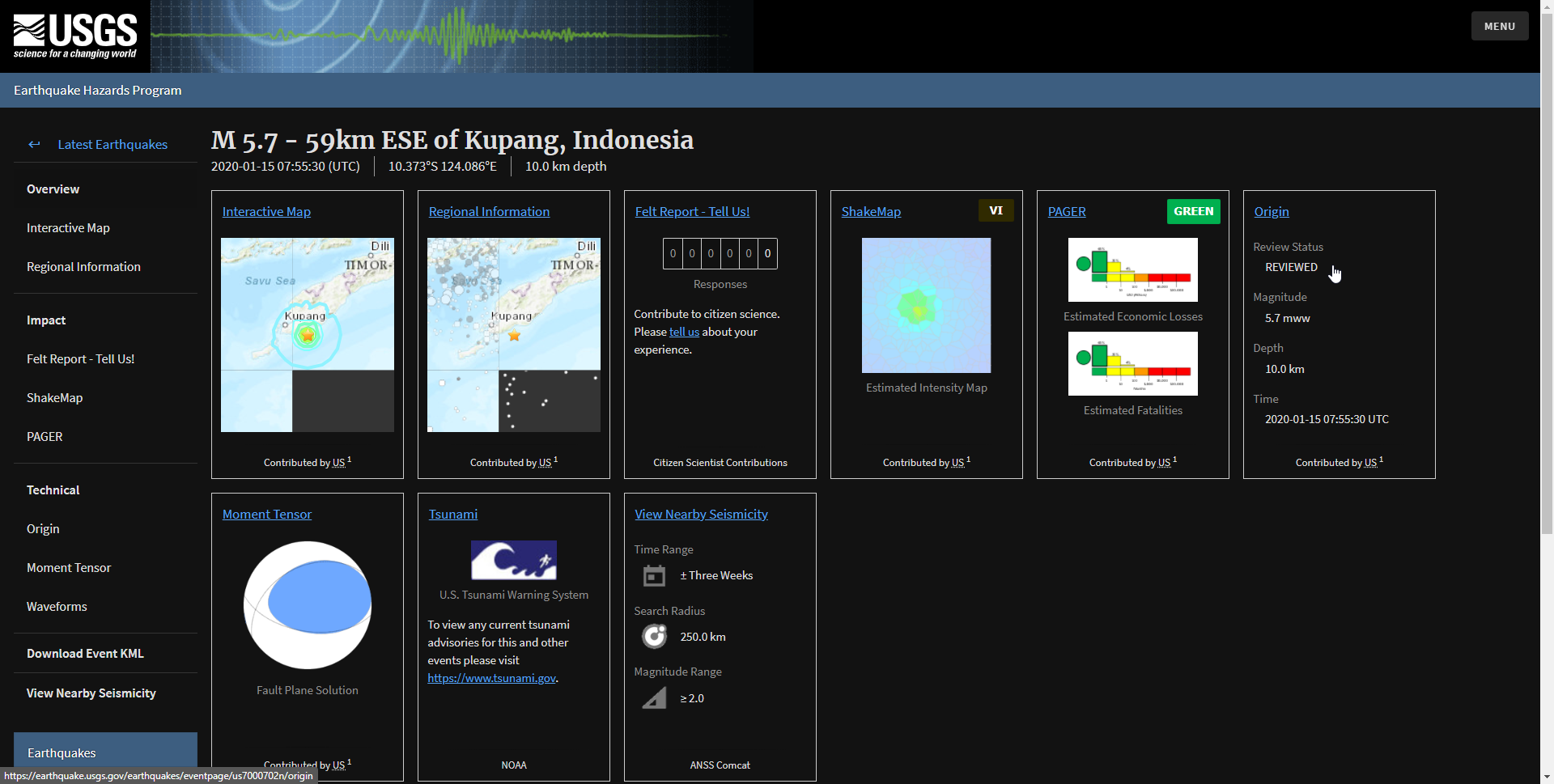


Figure 2.24: redirect to detailed page.

# Testing

# Conclusion

As seen in the Chapter 3: Testing above, the whole project works well.

Learnt:

* $key => $value

The link to GitHub file <https://github.com/Arouven/webservices>.