**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](#_Toc78314430)

[Chapter 2: Design and implementation 2](#_Toc78314431)

[2.1: Core functionalities 2](#_Toc78314432)

[2.2: Additional functionalities 3](#_Toc78314433)

[2.2.1: Headers 3](#_Toc78314434)

[2.2.2: How to use 4](#_Toc78314435)

[2.2.3: Scalable API 5](#_Toc78314436)

[2.2.4: XML API 8](#_Toc78314437)

[2.3: Client website 9](#_Toc78314438)

[Chapter 3: Testing 10](#_Toc78314439)

[3.1: Home page 10](#_Toc78314440)

[3.2: Regional page 13](#_Toc78314441)

[Chapter 4: Conclusion 15](#_Toc78314442)

[Annex A: myjs.js 16](#_Toc78314443)

[Annex B: display.php 17](#_Toc78314444)

[Annex C: database.php 18](#_Toc78314445)

[Annex D: credentials.php 19](#_Toc78314446)

[Annex E: myapi.php 20](#_Toc78314447)

[Annex F: index.php 21](#_Toc78314448)

[Annex G: detail.php 22](#_Toc78314449)

[Annex H: bookstore.sql 23](#_Toc78314450)

**List of Figures**

[Figure 2.1: function to add headers. 3](#_Toc78314451)

[Figure 2.2: code for how to use. 4](#_Toc78314452)

[Figure 2.3: push to array. 5](#_Toc78314453)

[Figure 2.4: WHERE and AND added to subquery string. 6](#_Toc78314454)

[Figure 2.5: full substring output. 6](#_Toc78314455)

[Figure 3.1: Home page. 10](#_Toc78314456)

[Figure 3.2: Testing of the date range. 11](#_Toc78314457)

[Figure 3.3: Testing of the data table sorting. 11](#_Toc78314458)

[Figure 3.4: Testing of the data table search. 12](#_Toc78314459)

[Figure 3.5: Testing of the world wide map. 12](#_Toc78314460)

[Figure 3.6: Testing regional page. 13](#_Toc78314461)

[Figure 3.7: Testing of parameters in regional page. 13](#_Toc78314462)

[Figure 3.8: Testing of regional map. 14](#_Toc78314463)

List of Acronyms

API - Application Program Interface 1

ISBN - International Standard Book Number 7

JSON - JavaScript Object Notation 1

php - Hypertext Preprocessor 1

SQL - Structured Query Language 6

URL - Uniform Resource Locator 5

XML - eXtensible Markup Language 1

# Introduction

In order to have some functionalities on the same page, a website is built to query a server and receive the JavaScript Object Notation (JSON) or extensible markup language (XML) file and display the required data. There will be a rest web service application program interface (API) using Hypertext Preprocessor (php).

# Design and implementation

## Core functionalities

The core functionalities developed on the API are also implemented in the client’s web site. They are as follows:

* Get books information by title
* Get books information by author
* Get books information by publication year
* Get books information by author and category
* Get books information by author and publication year

Rather than the traditional approach (SELECT \* FROM bookstore.books WHERE title=’harry potter’ ;), another type of approach has been used which will make the API more scalable and to work better with multiple query string. This will be explained further section 2.2.3: Scalable API.

## Additional functionalities

### Headers

In the case of the bookstore, either the books are found or not. But there is another one that can arise; bad request. To help programmability, status code, message, and total book in the database are prepended to the response. A function addStatusToData ($status, $statusMessage, $count, $data) was created (as show in Figure 2.1) to add the above before the data/response.

The status code and messages are as follow:

* 200 – Successful (represent book found in bookstore)
* 400 – Bad Request (represent wrong URL query)
* 404 – Not found (represent book not found in bookstore)

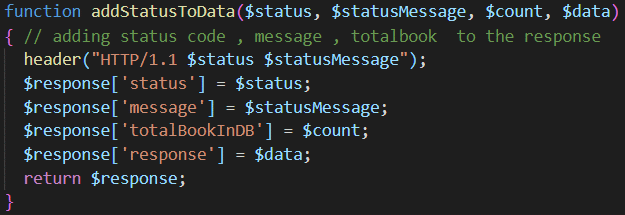


Figure .: function to add headers.

### How to use

If there is no query in the URL, it is assumed that the client don’t know how to use the API. Figure 2.2 shows part of the code to display the how to use page to the user. This will be further explained in Chapter 3: Testing.

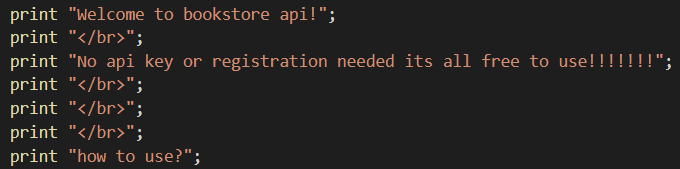


Figure .: code for how to use.

### Scalable API

As mentioned before, this will make the API more dynamic. The API will be programmed to allow any of its users to use it without any restriction in the number of query passed as request. Annex E: myapi.php has the full code for any reference.

The approach is as follows:

1. Get the main query – “SELECT \* FROM books”
2. Combine/concatenate it with the subquery (if any). A function was built to perform this task. It requires all the parameters of the Uniform Resource Locator (URL) to perform it task and return the subquery.

 $this->query = "SELECT \* FROM books" . $this->subquery($params);

**Exploring subquery ($params) method/function.**

The principle is that every query parameters should be prepend with the word “AND”. To achieve this, an array is created. All the query that required the keyword will be push to that array. Figure 2.3 shows two examples of parameters that should be prepended.

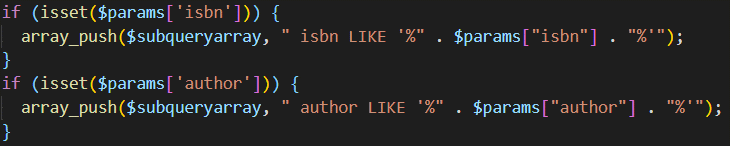


Figure .: push to array.

Then the next step is to add the “WHERE” keyword. This will be done only if there is something in the array. After that, a Foreach loop will do the job of adding the “AND” in front of each string in the array. The result of $subquery in Figure 2.4 with input author=harry&isbn=001 will be “ WHERE AND author LIKE '%harry%' AND isbn LIKE %001%”. The keyword LIKE will allow the user to insert part of the word to query.

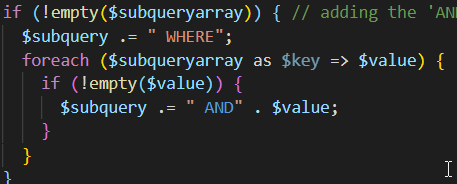


Figure .: WHERE and AND added to subquery string.

As the MYSQL syntax does not require any “AND” in front of “ORDER BY” and “LIMIT”, they were not push to array. For example, author=harry&isbn=001&orderby=rating&limit=5 will output structured query language (SQL) $subquery = “ WHERE AND author LIKE ‘%harry%’ AND isbn LIKE %001% ORDER BY rating LIMIT 5”. The last line of that function will solve the problem of the “WHERE AND” in front of the subquery by replacing it with “WHERE” and add the ending of SQL query as show in Figure 2.5. This will lead to output “ WHERE author LIKE ‘%harry%’ AND isbn LIKE %001% ORDER BY rating LIMIT 5;”

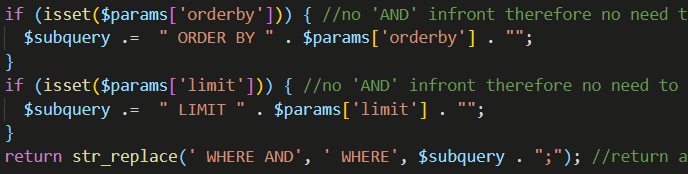


Figure .: full substring output.

Therefore,

 $this->query = "SELECT \* FROM books" . $this->subquery($params);

Will contain “SELECT \* FROM books WHERE author LIKE ‘%harry%’ AND isbn LIKE %001% ORDER BY rating LIMIT 5;”

Using this type of approach, the following can be accomplished:

* Search for word that contain some characters in title, author, International Standard Book Number (ISBN), category, language, and year published.
* Order the result by author, category, etc.
* Limit the result of the search to x number. Where x is a number define by the user.
* Above all, the user is now unlimited with the column to query. He/she can query more than one for example author and category and much more.

The query is then executed and return in a form of array. This array will be converted to either XML or JSON and outputted to the user.

### XML and JSON format

#### JSON format

Figure 2.6 show the code that will output the data in JSON format. Header is added and php already has its own function to output in JSON (json\_encode). This will take the array outputted from the SQL query and convert it to JSON format.

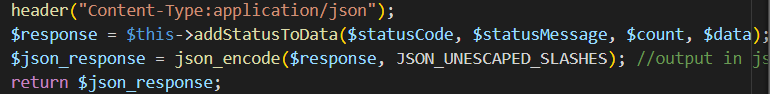


Figure .: code to output in JSON format.

#### XML format

XML on the other hand is more complicated as there is no inbuilt method to encode in XML format. But the principle stays the same. A user-defined function should be created to perform this task. Figure 2.7 shows how the user-defined will look like.

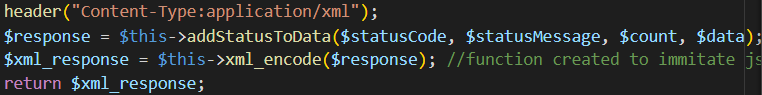


Figure .: code to output in XML format.

In Figure 2.8, the XML header is added together with the first element. Then the array will be retrieved from the SQL query will be converted to XML with bookX tag(s) (X start from 0 to total record in array) and inserted between the bookstore tags.

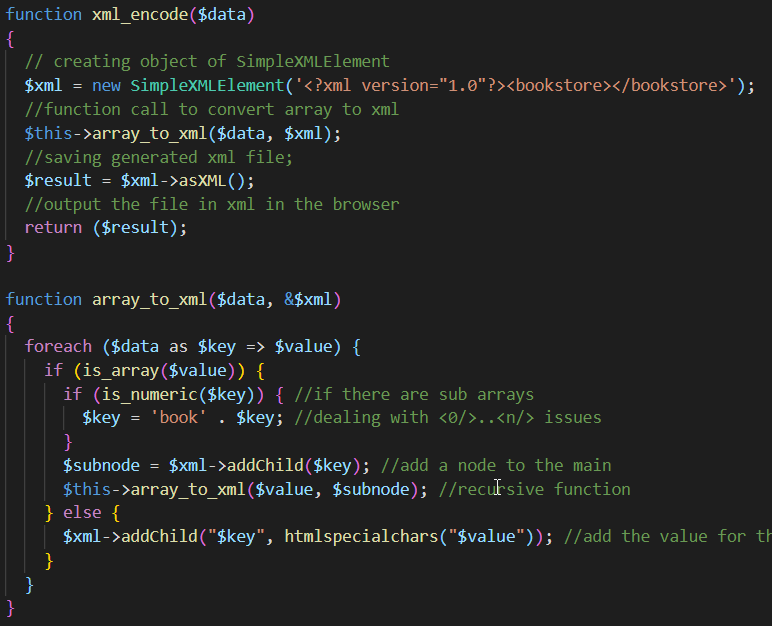


Figure .: user-defined function xml\_encode

## Client website

The client website support both XML and JSON responses.

# Testing

## Home page

The page contains the elements discussed before.

|  |
| --- |
|  |
|  |
|  |
|  |

Figure .: Home page.

Testing of the date range: the js outputted the invalid range if the start date exceed the end date.

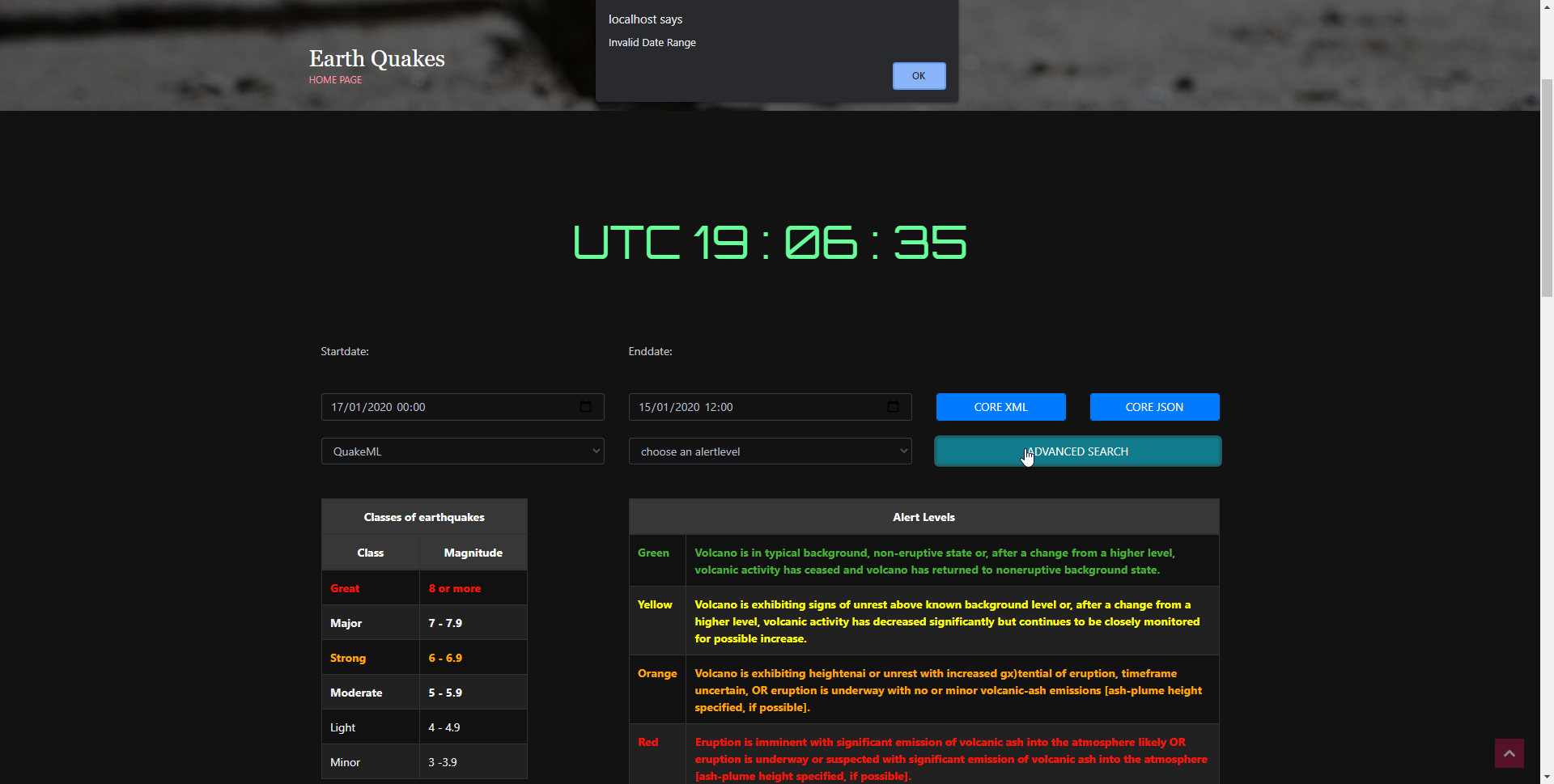


Figure .: Testing of the date range.

The sorting was tested with different columns.

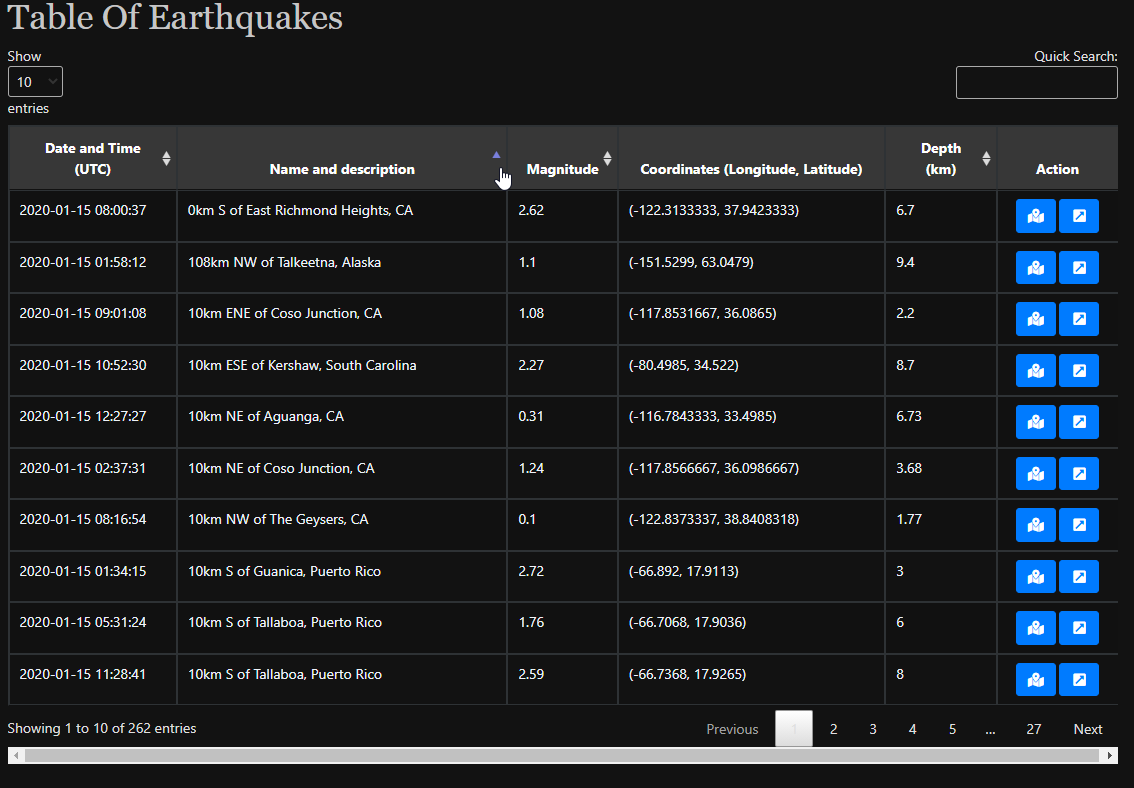


Figure .: Testing of the data table sorting.

The testing of the on press the quick search

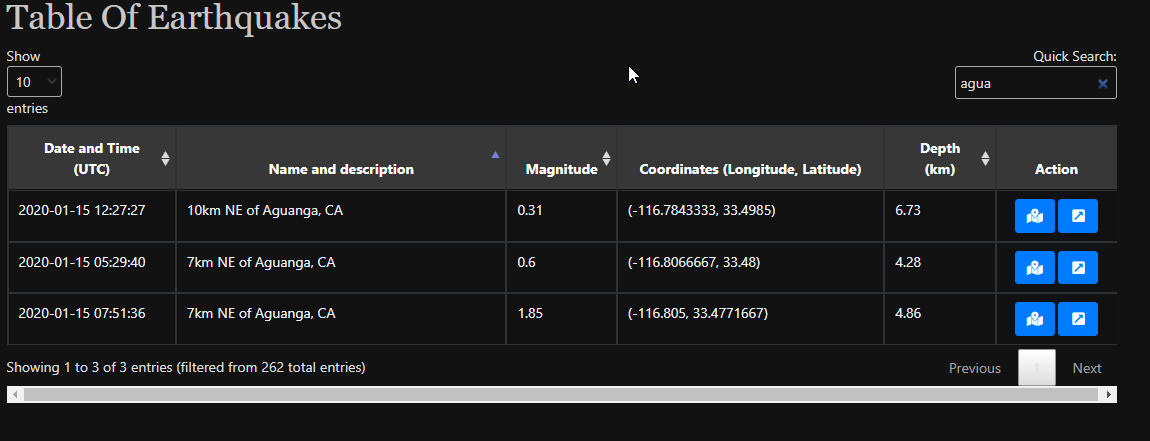


Figure .: Testing of the data table search.

The worldwide map was tested the zoom works well and the movement of the map too.

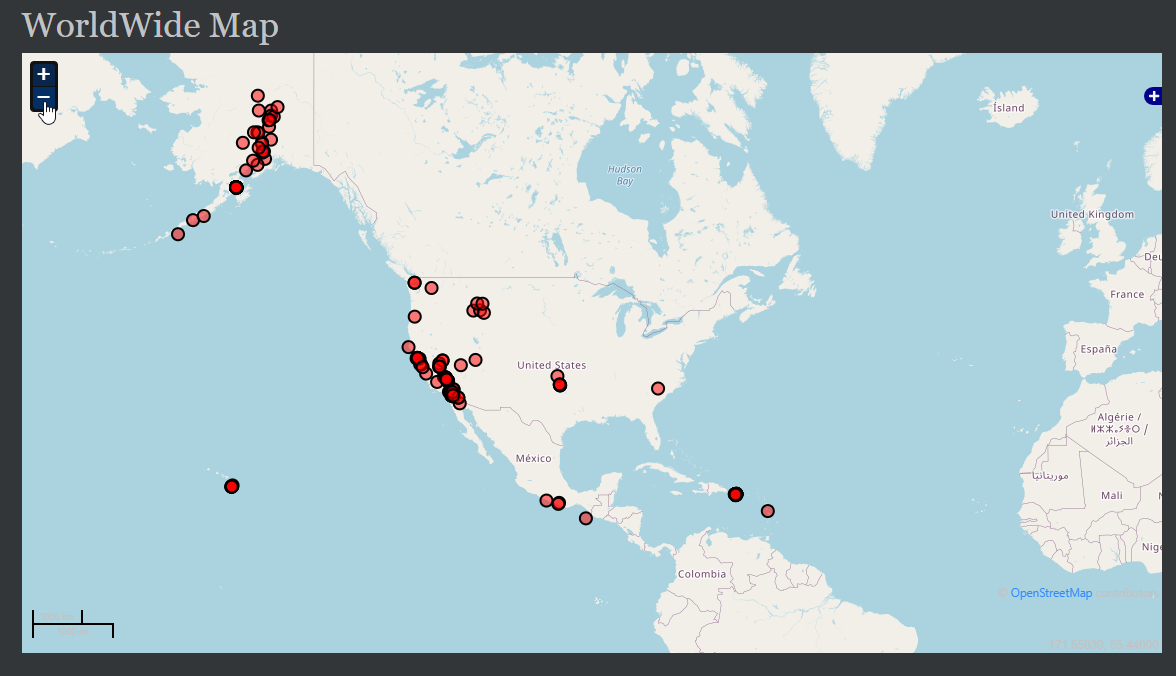


Figure .: Testing of the world wide map.

## Regional page

The page contains the elements discussed before.

|  |
| --- |
|  |
|  |
|  |

Figure .: Testing regional page.

Testing of button get location and other parameters

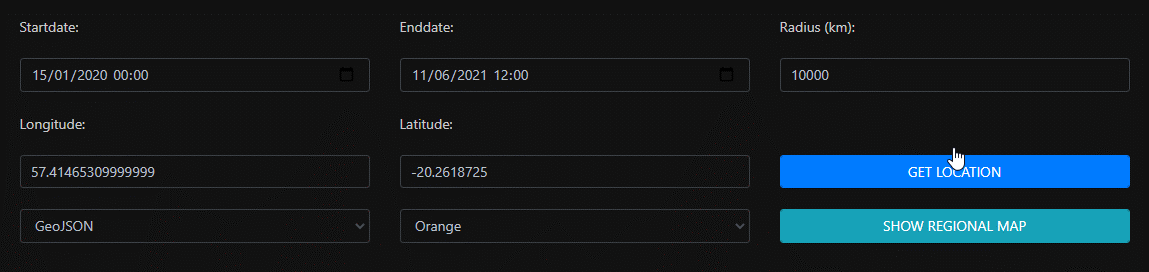


Figure .: Testing of parameters in regional page.

The map works well as it follow the same worldwide map above.

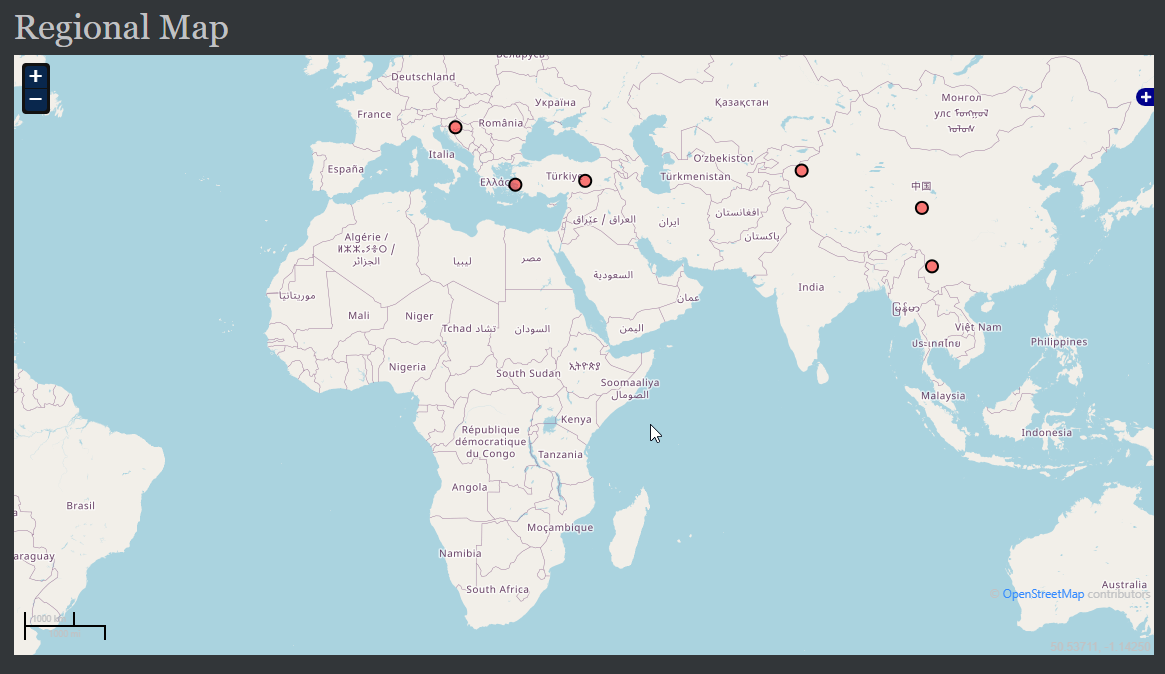


Figure .: Testing of regional map.

# Conclusion

As seen in the Chapter 3: Testing, the whole project works well. As an improvement, a dot on the regional map could be inserted to indicate the location of the place the radius of earthquakes.

The link to GitHub file:

<https://github.com/Arouven/webservices>

Test it yourself:

<http://contacttracing.great-site.net/assignment_1_EarthQuakes/pages/index.php>

# Annex A: myjs.js

# Annex B: display.php

# Annex C: database.php

# Annex D: credentials.php

# Annex E: myapi.php

# Annex F: index.php

# Annex G: detail.php

# Annex H: bookstore.sql