|  |
| --- |
| Faculty of Applied Sciences and Technology |
| **XML/JSON Data Processing** |
| ITC5202 - Project |
|  |
| **Bikal Thapa Magar & Merin Joe Chandy**  **Submission Date** |
| **4/18/2022** |

|  |
| --- |
| This document explains how to process XML/JSON data …………………………. |

Table of Contents

[Question 1: Use JavaScript/Ajax to process XML data 2](#_Toc519253004)

[Question 2,3,5,6: Use JavaScript/jQuery to process JSON data 4](#_Toc519253005)

[Question 4 and 7(Bonus) : XSLT and XPath 9](#_Toc519253006)

[Summary 12](#_Toc519253007)

# Question 1: Use JavaScript/Ajax to process XML data

(Describe the major steps for designing the JavaScript function(s), how you test this program, add some screenshots of the output )

The first step is to get the data from the xml file. For this we used the .ajax() function of jQuery.

After we get the xml data from the file in our script, we then need to filter out and grab the only data that we require. In this case we need name of the country, it’s capital, unemployment, and the total GDP.

Before extracting the data, we need to know the structure of the dataset. The country name, unemployment, and the total GDP were on a separate tag, but this was not the case for the capital city. There were multiple cities under the country tag and each city has an id and the id of the capital city is tagged to the capital attribute of the country element.

First, we grabbed all the elements with country tag using the **find()** method and then looped through each of them using the jQuery **$.each()** method.

Then, we extracted the country name, unemployment, and the total GDP using the combination of find() and text() method as:

**$(country).find(‘unemployment’).text();**

For, the country name we used .first() method as well after the find() as there were multiple tags for name tag within a country tag.

**$( country).find(‘name’).first().text();**

And for the capital city, we first grabbed all the elements with city tag and then looped though each of them and checked whether its id matched with the capital id. If the id matches, then we have the capital city and grabbed its name.

Finally, to display the data, inside the loop of countries, we prepared a row for each country’s name, capital, unemployment, and total GDP, accessed the html table using the id selector and pushed the row using the **.append()** method.

With the development of each step, we printed the output in the console to check if we are getting the desired result before proceeding to the next step.

Table

Description automatically generated

# Question 2,3,5,6: Use JavaScript/jQuery to process JSON data

(Describe the major steps for designing the JavaScript function(s), how you test this program, add some screenshots of the output)

Question 2

The major step for this is getting the user input and display the country details.



The .**click**() function fetch the input from the user and on success condition, it checks the input and displays the details based on the country name.

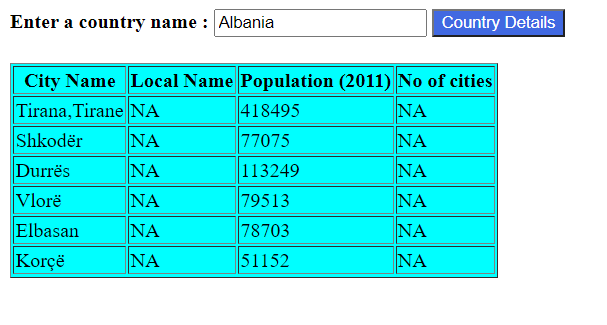
The conditions check for Whether the country has province or not.

If province is present, the table will be displayed as follows: -



Details of countries without a province will be displayed ad below:-

(Note: The table header is changed based on the condition.)



The code also checks Whether a local name or cities are available, if not it will be displaying the data as “NA”.

Question 3

The major steps include getting the user data, getting the news from news API and displaying it.

First, we designed the jQuery to handle the click event of the form button using the .**click()** method. Inside this, we grab the input from the user and pass it to another function to fetch the data.

To fetch the data using the API, we first created the URL by appending the country code entered by user. We then passed the URL in the fetch API and received the headlines data in JSON format.

We then designed another function which takes this JSON data as input, extracts the required information, and displays it in the browser window. We first clear the existing news headlines using the .**empty()** method on the wrapper tag element. After that, we grab the only news headlines data from the JSON data and loop through each of them. From each of the news, we extract their source, headlines, URL, image, and published date. After we have all the required data, we prepared the output to be shown and append it to the wrapper element. Some of the news does not have image, so we also checked if the image URL is null or not and display it accordingly.

We tested this program by printing the data in the console on each step, or when the output was not as expected to debug the code and often used debugger.

For the bonus part, we implemented the translation feature using a button to translate the headlines to English. When the button is clicked, a JavaScript function collects all the headlines and using [Just Translated](https://rapidapi.com/lebedev.str/api/just-translated/) translation API converts the headlines to English and updates them in the browser.

Graphical user interface, website

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, chat or text message

Description automatically generated

Question 5

The steps for solving this question were also like the previous ones i.e., call the API, get the response, extract data from response and show it in the browser. One Challenge we faced while solving this question was to get the full names of the neighboring country as the API was only providing 3 characters ISO country codes. For this we had to call API to get the full name of each of the neighboring country.

Initially we tried to get the names of each neighboring country, append it to the output variable and render it in the browser. While this method worked for those countries that had few numbers of neighboring countries, it was relatively slower for the countries having many neighboring countries, for instance, China.

So, we came up with an alternative method. In this, we first get the country data and render it in the browser except for the neighboring countries. After the country data is rendered, we then call the API to get the full name of each of its neighboring country and render it in the browser as we get the data.

A picture containing graphical user interface

Description automatically generated

Text

Description automatically generated

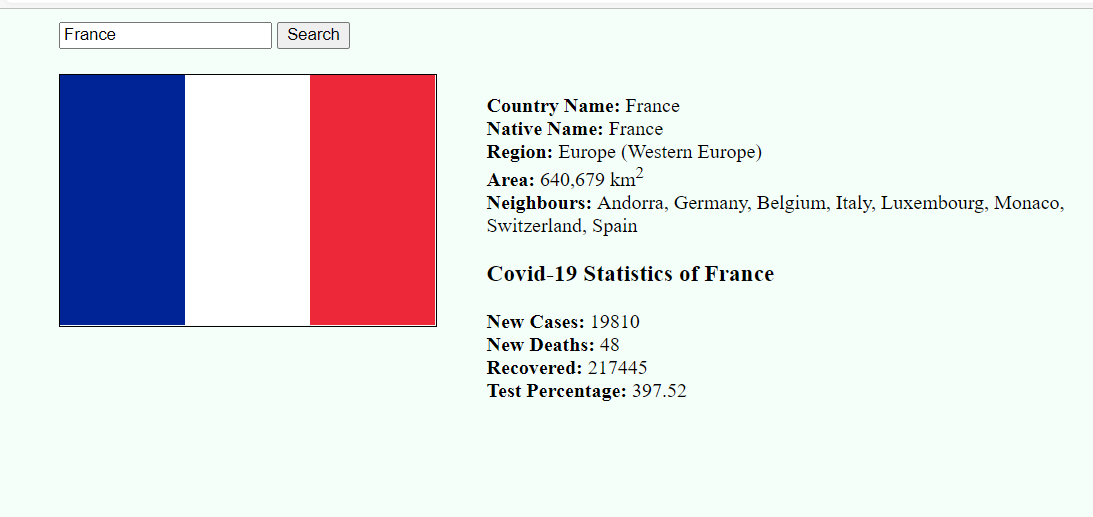
Question 6

The requirement of the project was to add additional data to Q5, by using a related API.

The API we used is for getting the COVID-19 statistics of countries.

We created an async function for getting the Country details where we invoked the API by passing country name and code.

When the user inputs a country name, based on the country name and country code, it will append the covid static data we tried to fetch, along with the output of Q5.



# 

# Question 4 and 7(Bonus) : XSLT and XPath

(Describe the major steps for designing the XSLT. How did you use XPath in the XSLT?

Add screenshot of the XPath testing and the output of XSLT)

Question 4

The layout and design of the output was like that of question 1, the only major step required for this question was to get the correct data using the XPath expressions. To check the XPath expression, we used the XPath test tool from [freeformatter](https://www.freeformatter.com/xpath-tester.html) website. Also, the original xml file was large, so we used a subset of the country data and used it to test the XPath expression in the XPath test tool and to view the output of our XSL file as well.

Extracting country name, unemployment, and total GDP was straight forward, we looped through each of the country tag using <xsl:for-each> and retrieved the required data.

Retrieving the capital city was little tricky though. Some of the cities were directly under the country tag whereas some of them were under the province tag. Likewise, the id of the capital city was present as an attribute of the country tag, and we had to compare the id of each city with this. So, to retrieve the capital city, we first looped through each of the city under the current country. Then we accessed the country tag of that city using the ancestor::parent\_node method. From there we compared the capital attribute of the country with the id of the city using <xsl:if test=”expression”> and displayed the first name of the city as it had multiple names.

Graphical user interface, text, application

Description automatically generated

Table

Description automatically generated

Question 7

The major steps involved to solve this question were: first to get the input from the user, second to extract the data of the country provided by the user, and third to show the data in proper format.

We did not find any way to pass the input from the text field to the xml so we opted for the drop-down menu. In the dropdown list, we populated the dropdown data with the country name, and populated their values with the entire data to be shown for that country. Now that we have a way to get input from the user and the data to be shown, we used few lined of JavaScript code to know the country selected by the user. Once we have the country, we get the values of that selection and show it in the browser.

Many countries did not have provinces. In such case we used the XPATH **not** function to check whether province exists or not. If there is no province, then we have shown the country data itself.

Likewise, there were no data for many countries and provinces. By default, blank cells were shown in such cases. We used the <xsl:choose>, <xsl:when>, and <xsl:otherwise> tags to show ‘-’ in such scenarios.

Graphical user interface, application

Description automatically generated  
Table

Description automatically generated

# Summary

We divided the task so that a single question was done completely by one member so that they can get complete knowledge of it.

During this project we got to learn a lot of stuffs specially regarding consuming the API using AJAX. For most of the questions, we used the jQuery ajax function. Though the code for the fetch API was shorter, the jQuery ajax function was much easier to use and understand.

Furthermore, we learned how to consume more than one API in a single program and how to use the async/await in asynchronous programming.