V3.0

WESTERN UNIVERSITY
CS 2212A
Introduction to Software Engineering

Project Description
Fall 2020-2021

1. Introduction

For this project you will specify, design, and implement a system that allows for retrieving COVID-19 infection related data for one or more selected countries, computing different statistics on these retrieved data, and rendering the computed statistics on a map. Such an example statistic can be *confirmed cases per capita*.

In order to compute the different statistics, you will need to retrieve data related to total confirmed Covid-19 cases to date, a country's population, Covid-19 confirmed cases for women, or Covid-19 confirmed cases for men. You are free to choose and retrieve other different types of Covid-19 related data in order to perform your own types of analyses. The source of the data can be a file you have downloaded or any site which can be accessed over the Internet and returns an XML or JSON response. Examples of such sites where such data can be retrieved are given in the section "Data Acquisition" below. You are free to pick other web sites that offer data that suit the needs of your analyses.

Some example statistics which you can compute for the needs of this project can be *a)* the total confirmed cases for a given country or a list of countries, *b)* the total confirmed Covid-19 cases per capita for a given country or a list of countries, *c)* the total confirmed Covid-19 cases per sex (males or females) for a given country or a list of countries. Your system and your design should allow for new statistical analyses to be added easily without affecting the design or the implementation of existing ones.

Once the statistics are calculated they have to be displayed on the map. The way you display the results is up to you, but it has to be intuitive, that is the statistics results are easy to understand, visualise and interpret. For example, the statistics for a country can be displayed by drawing a circle, on the appropriate point on the map, where the area and color of the circle are proportional to the statistical values computed. More details on the rendering of the results can be found in the section "Data Rendering" below.

The system will have a main User Interface (UI) where all actions are performed. An example layout of a possible main UI and its different parts is depicted in Figure 1 below.

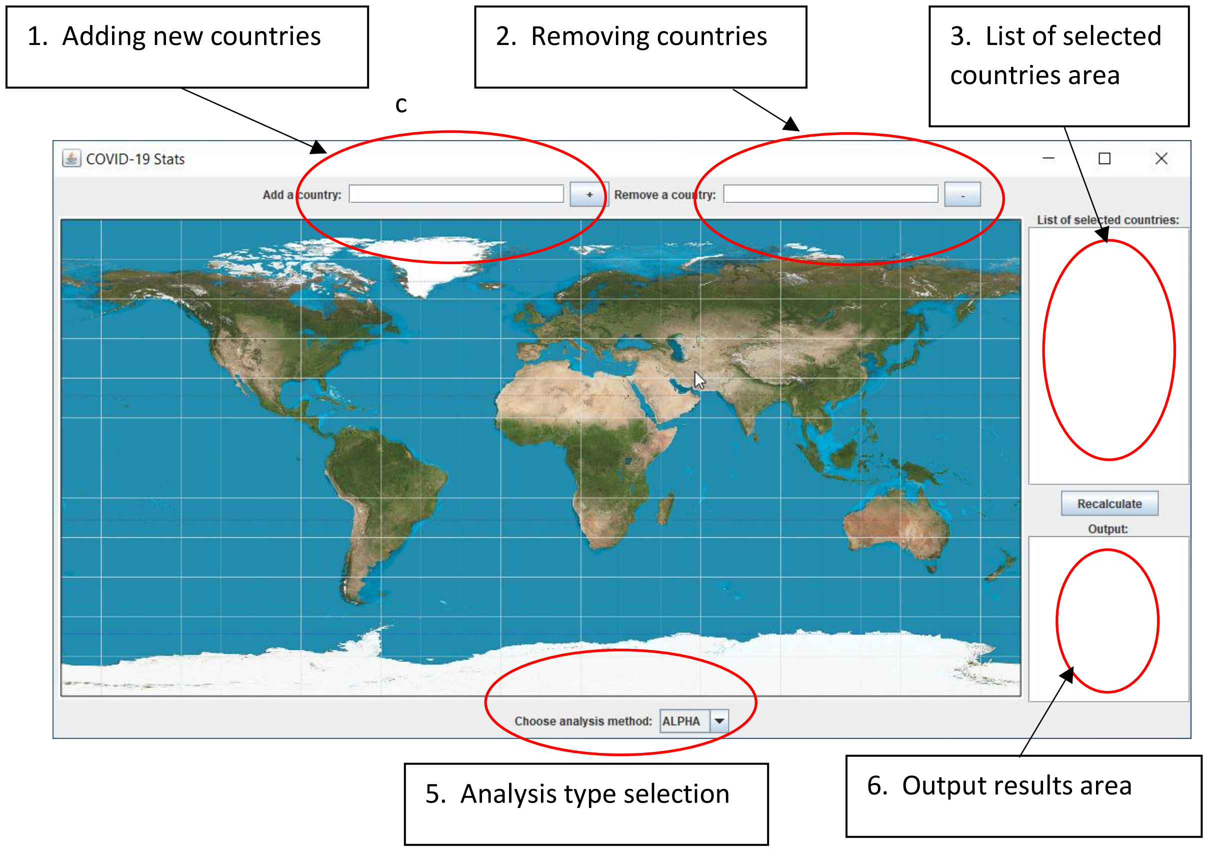


Figure 1. Sample Main UI for Covid-19 Statistics System

2. Use Cases

The uses cases for your system are provided below:

UC1. The user logs into the system

When you start your system, the users are greeted with a login window or form, where they can input their username and password. If the combination of username and password is not correct or no such user exists in the system's database, a pop-up window or a notification in the form will notify them that there is an error with the provided credentials and the application will terminate. You can store the username-password pairs either in a database, or in a text file. When using a file, the format can be plain text , JSON or XML. If the username-password combination is correct, then the main UI of the application is displayed.

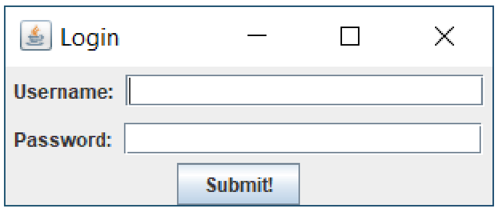
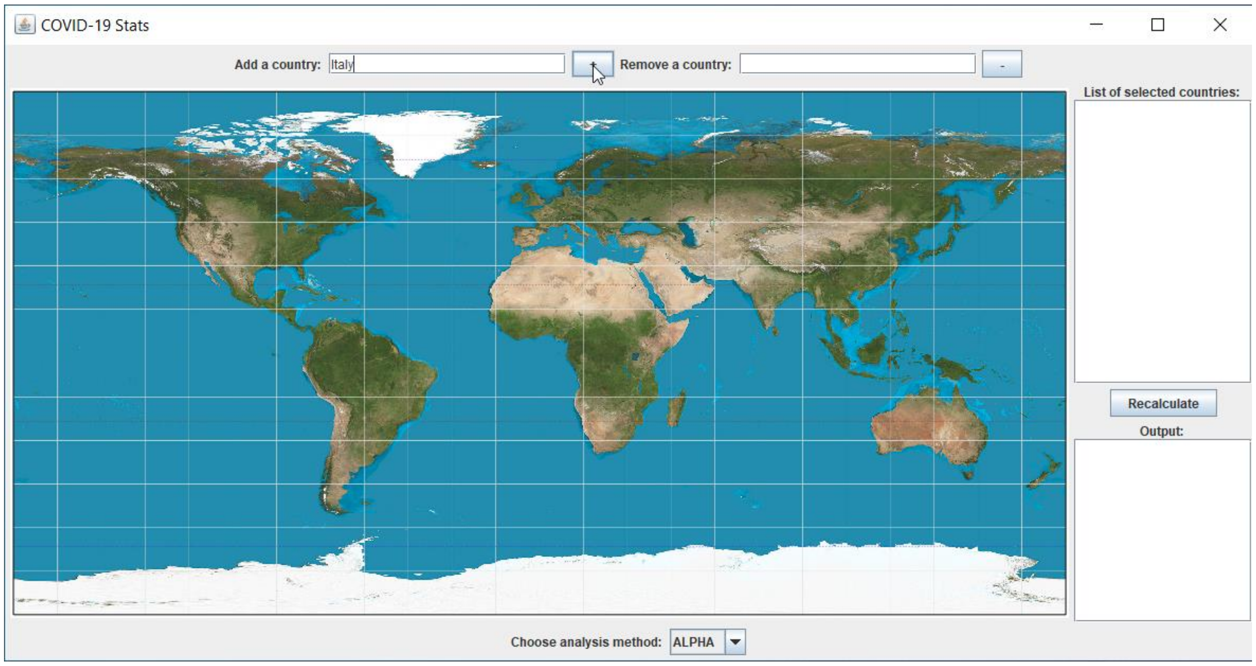


Figure 2. Sample Login panel

UC2. Adding new countries to the list of countries to perform analysis of

The users can add new countries in the list of countries they want to perform analysis of. For this use case, the users will either type the name of the country they want to add to the list, or select the country from a drop-down menu. The user can then select the "add" (or the "+") button to add the country to the list of countries to perform analysis of. Once the user presses the "add" or the "+" button, the system checks whether this country is in the list of countries the system knows about. If the country is not in the list of known countries, then an error message is displayed. For example, a misspelled country name will result in such an error. If the country selected is in the list of available countries, it is added to the list of countries, on which analysis is to be performed. The county's name must also be displayed on a UI panel along with all the other so far selected countries.



Add a country: Italy

Choose analysis method: ALPHA

Figure 3. Adding Italy to the list of countries to perform analysis on

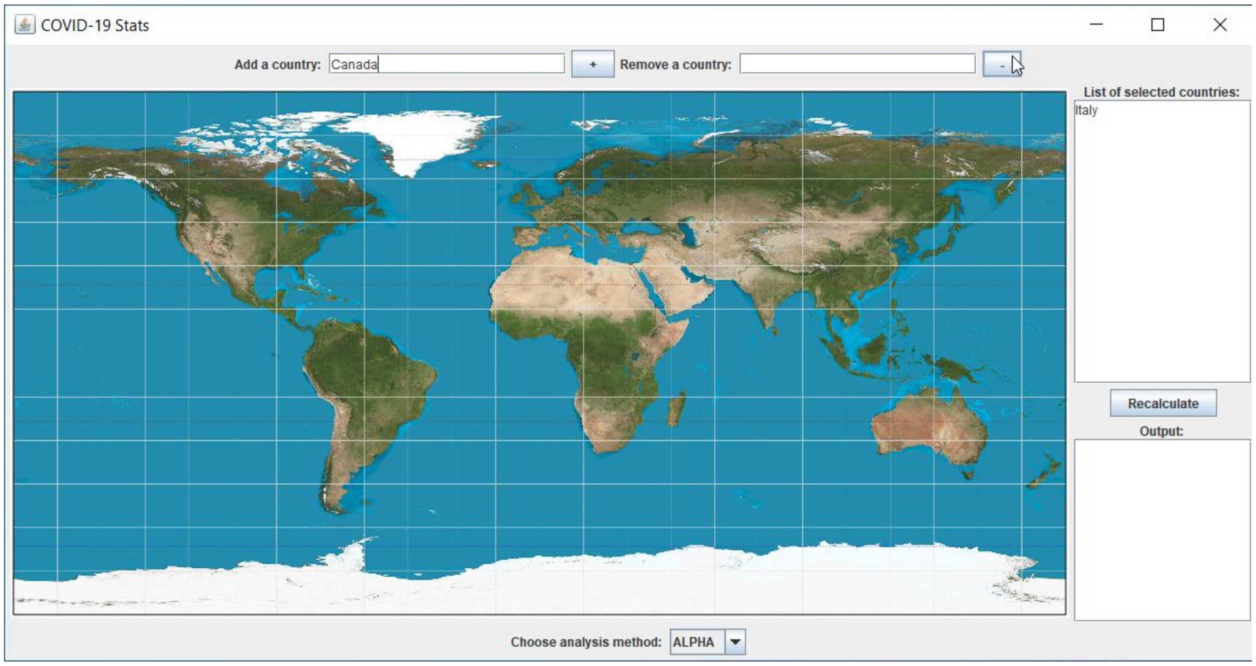


Figure 4. Adding Canada to the list of countries to perform analysis on

UC3. Removing countries from the list of countries to perform analysis of

The user can remove countries from the list of countries they want to perform analyses of. For this use case, the user will either type the name of the country they want to remove from the list or select the country from a drop-down menu. The user can then select the “remove” (or the “-”) button to remove the country from the list of the countries to perform an analysis on. Once the user presses the “remove” or the “-“ button, the system checks whether this country is in the list of countries the system knows about. If the country is not in the list of known countries, then an error message is displayed. Same as before, a misspelled country name will result in such an error. If the country selected is in the list of available countries, it is removed from the list of countries, on which an analysis is to be performed on. The updated list of countries is displayed accordingly. If the country selected to be removed was not on the list in the first place, a message is displayed to the user indicating that the country they want to remove from the list was not in there in the first place.

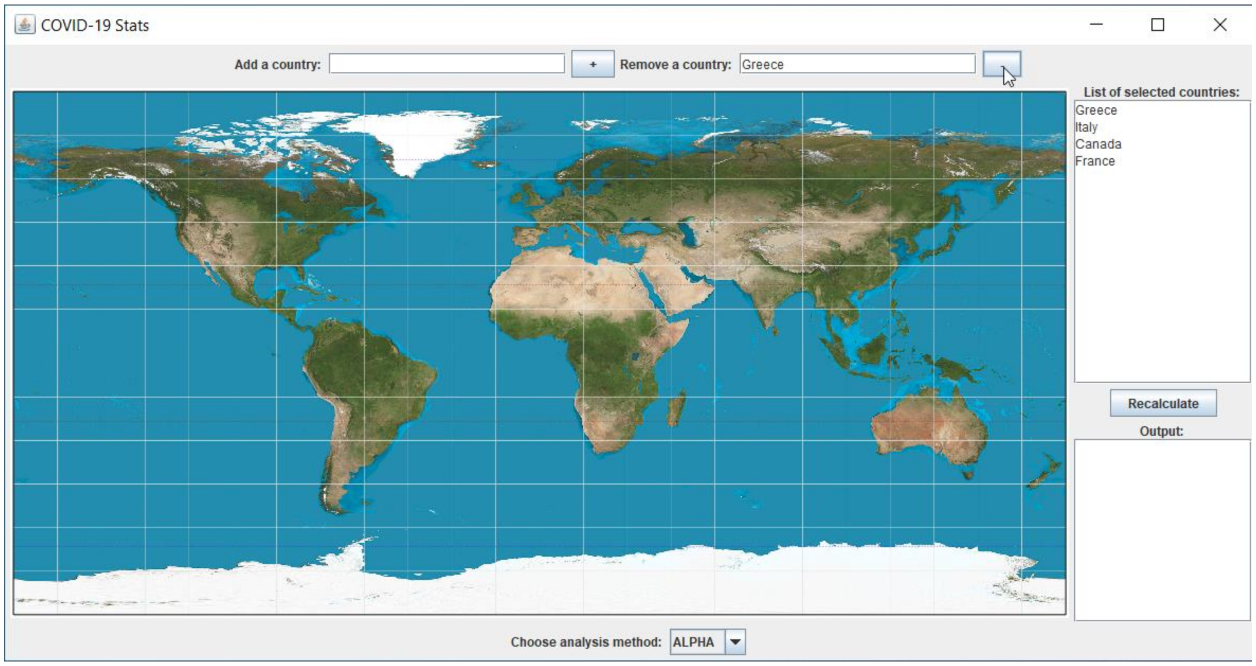


Figure 5a. Removing Greece from the list of countries to perform analysis for

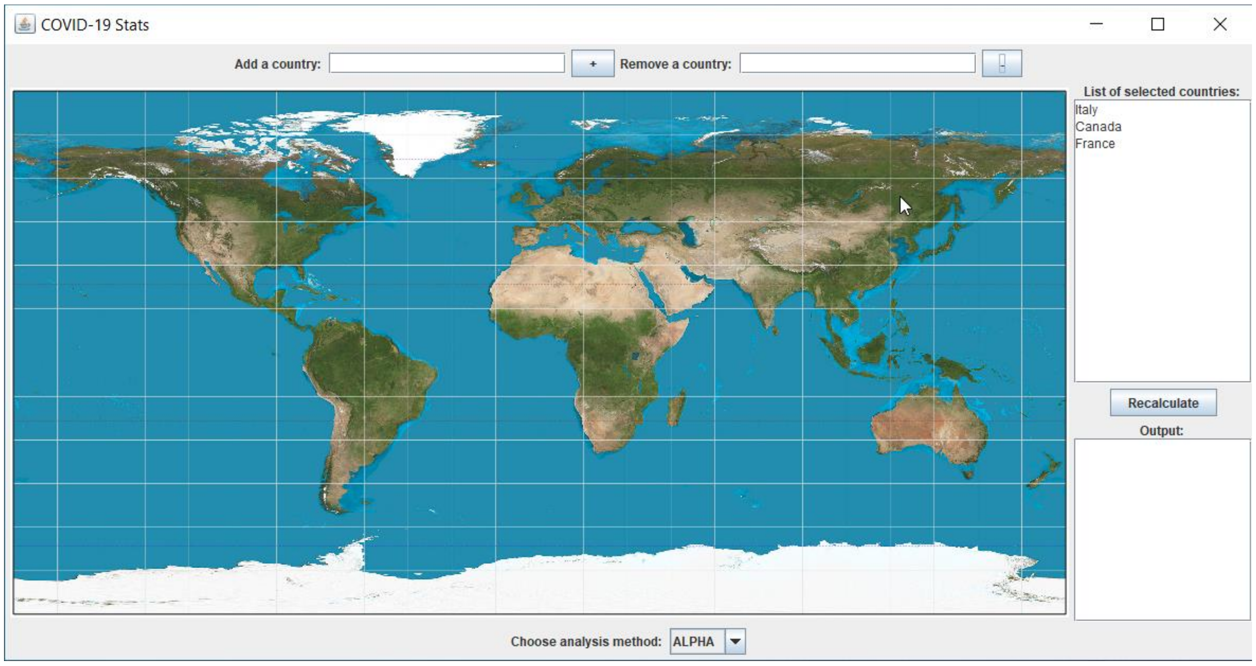


Figure 5b. Updated list of countries to perform analysis for

UC4. Selecting the analysis type to be performed

Once the user has selected the countries to perform analysis on, they can select the specific type of analysis they want to perform from a drop-down menu of available analyses. You should implement 4 different types of analyses. Your design and implementation has to be modular. For example, the data required for each analysis type can be obtained in a modular way (i.e. by separate "reader" classes which read the specific data from the appropriate web site) and the analyses themselves have to be performed in a modular way, that is, each type of analysis has to be performed in its own class.

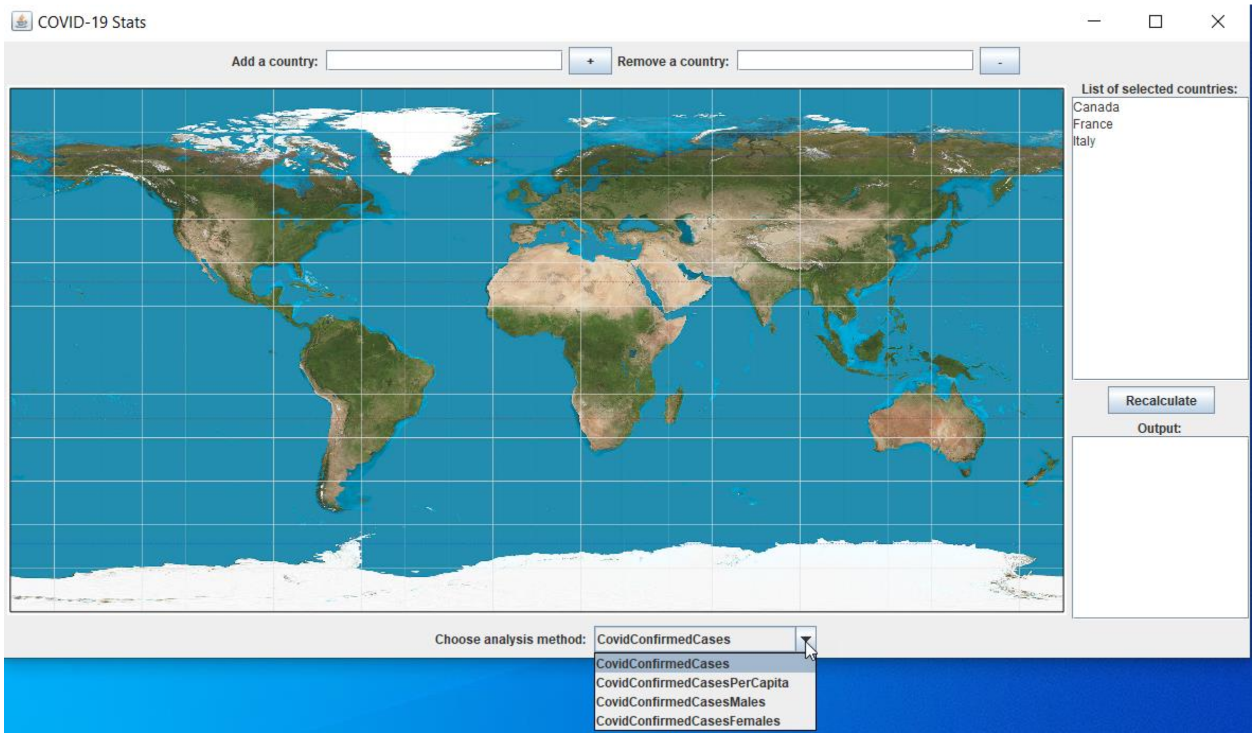


Figure 6. Selecting the Analysis Type

UC5. Performing the analysis

Once the user has selected the type of analysis they want performed, they can initiate the analysis by pressing the "Recalculate" button. This will initiate the identification and retrieval of the individual pieces of data required for the type of analysis selected. Once all the required individual pieces of data have been collected, the computation of the specific analysis commences. For example, if we want to calculate the Covid-19 confirmed cases per capita, we need two pieces of data *a)* the total number of confirmed cases for a given country, and *b)* the total population of that country. Once we have collected these two pieces of data then the analysis will compute the *total-cases/total-population* ratio and return the result. If some, or all of the data required for the computation of the selected type of analysis are not available, a message is displayed to the user. If the analysis proceeds correctly, then the results are returned so that they can be displayed (see UC6). The same analysis has to be performed for all selected countries.

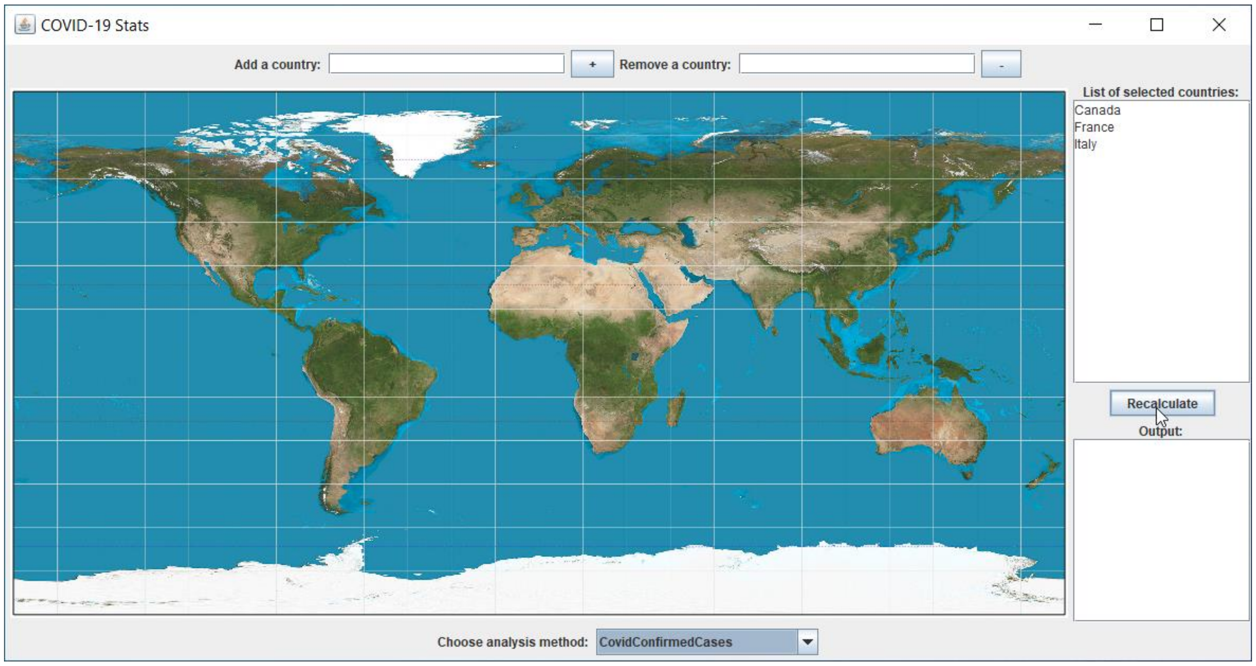
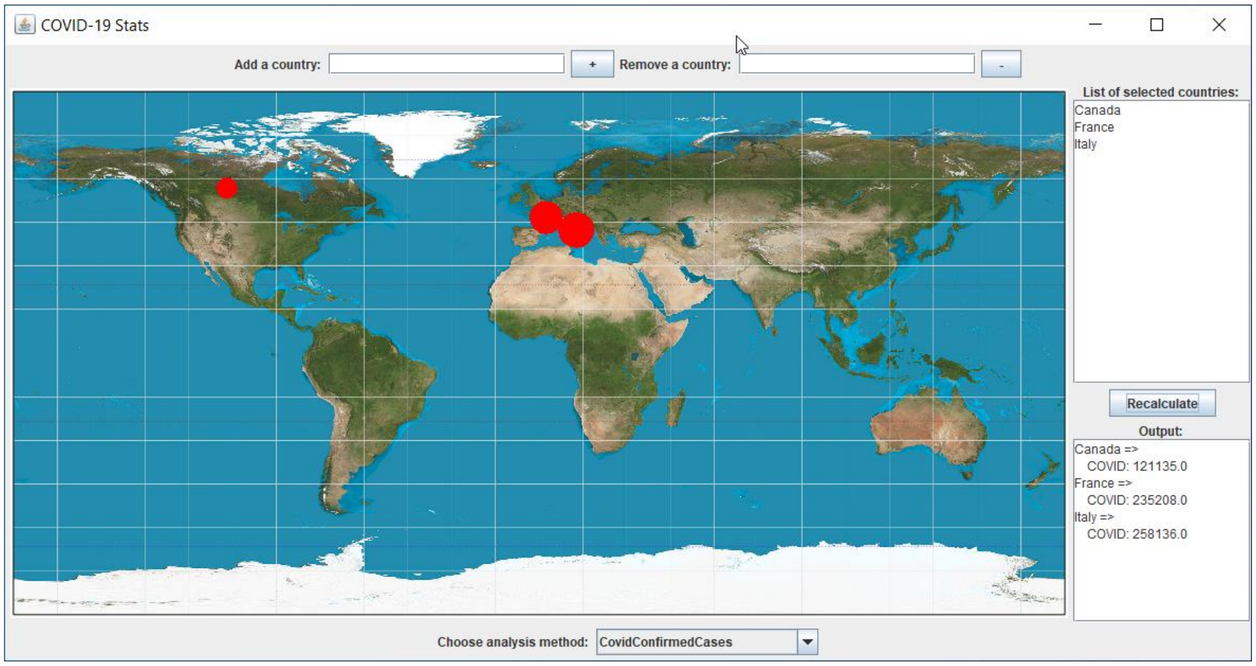


Figure 7. Triggering the Analysis

UC6. Displaying the results

Once the selected statistical analysis is completed and its results have been calculated, then the system renders the results on the map and displays them in the output results area (see Figure 1). It is your choice how to display the results. One way for example is to draw a circle on top of each selected country. The size of the circle should be proportional to computed values. For example, if you have selected three countries and the type of analysis is “Covid-19 cases per capita” and you get the following results Country A: 0.015 cases/person Country B: 0.01 cases/person and Country C: 0.005 cases/person, then the circle for Country A can have the max selected area (say the one which corresponds to a radius of 0.1cm), the circle for Country B can be 0.66 of the *area* of Country A (i.e. 0.01/0.015), and for Country C can be 0.33 of the *area* of Country A (i.e. 0.005/0.015). The system should be able to render the data in different colors (or even shapes if you choose to) depending on the value of the analysis data. For example, if the confirmed cases per capita statistic has a value greater than 0.1 case/person then the circle can be red, otherwise can be blue. You are free to choose the threshold values per type of analysis, the display color and the display format as long as it is intuitive and ergonomic (i.e. not just displaying the results in text using very small or overlapping between countries characters, or having infinitesimally small circles for countries with very low population). For information on rendering the results on the map, please see Section 4 “Rendering the Results” below and the Appendix section.



List of selected countries:
Canada
France
Italy

Choose analysis method: CovidConfirmedCases

Output:
```
Canada =>
   COVID: 121135.0
France =>
   COVID: 235208.0
Italy =>
   COVID: 258136.0
```

Figure 8. Displaying the Results

3. Data Acquisition

Your statistical analyses will require data to either be retrieved from one or more remote sites by issuing an http GET request from your Java application, or be fetched from a local file you have already downloaded. An example snippet of Java code on how to issue an http request and obtain Covid-19 confirmed cases from the https://api.covid19api.com site is given in Appendix I.A below.

- For Covid-19 confirmed cases (for issuing an http request to obtain a json file) https://api.covid19api.com/total/dayone/country/%s/status/confirmed, where %s denotes the name of the country we want data for.
  For example, by issuing an http request to https://api.covid19api.com/total/dayone/country/italy/status/confirmed you get all Covid-19 cases in different time periods.

- For population by country (file to download for local access) https://github.com/samayo/country-json/blob/master/src/country-by-population.json

- For population by country and sex (a web site with several files to download for local access) https://population.un.org/wpp/Download/Standard/Population/

- For Covid-19 cases per sex https://globalhealth5050.org/covid19/age-and-sex-data/ (data not for all countries)

- JSON format overview https://en.wikipedia.org/wiki/JSON and https://www.tutorialspoint.com/json/index.htm

You are free to identify alternative sources of data and compile your own types of analyses. Make sure though that the design of your system is modular. An example of modularity is that you have different "reader" components for different data sources.

4. Rendering the Results

The identification of the point where you need to render the analysis results (e.g. the center of a circle, should you choose to use this method of displaying the results – see UC6), can be performed using a simple formula. The formula takes the latitude and longitude of a country as parameters and produces the appropriate *(x,y)* coordinates on the supplied image of the world map. The *jpeg* world map is given to you, along with the *.csv* file listing the longitude and latitude coordinates of all countries. For your convenience, the programming logic that maps the latitude and longitude coordinates of a country to the *(x,y)* coordinates of this country in the *jpeg* map, is listed in the Appendix I.B below in the form of Java snippets. Also, the logic where you draw a circle on the *jpeg* map is also given to you in Appendix I.C in the form of a Java code snippet. See also Figure 7 above. The Java libraries you may need to perform such renderings are listed in Appendix I.D.

5.Design and Implementation Notes

The design and implementation of your system has to exhibit the following properties:

1. The architecture has to be modular and conform to one or more of the architectural styles you learn in class.
2. The architecture has to exhibit the properties of high cohesion and low coupling. Here high cohesion means that packages are designed to contain classes that perform a specific functionality of the system (e.g. a package that contains all classes that perform the different analysis strategies).
3. Your code must implement correctly several design patterns. The design patterns which are applicable to this system are: *Proxy*, *Façade, Chain of Responsibility*, *Strategy*, *Observer*, *Singleton* and, *Factory Method*. For example, the Proxy design pattern can be used for login services, the Façade design pattern for coordinating activities between the UI and the backend services (e.g. retrieving data, rendering images, checking for proper country names etc.).
4. You may create your project as a *Maven* project or convert it afterwards (see https://crunchify.com/how-to-convert-existing-java-project-to-maven-in-eclipse/). This will allow you to automatically download and install in your project any external libraries. An example pom.xml file which is needed for such a Maven project can be found in the Resources/Project folder in OWL.
5. For your collaboration you are expected to use bitBucket as your source code repository where you will be committing regularly your code changes and any supporting documents. We will provide bitbucket accounts for each team. For communication with your fellow team members you are expected to use Microsoft Teams. You should provide a trace record of the code commits each one of you performed throughout the term.

6.Tools

You can use the following tools to proceed with the deliverables of your project:

Drawing UML Diagrams:

1. UMLet (https://www.umlet.com/) This is the best choice

2. MS-Visio (download the UML stencils) https://support.microsoft.com/en-us/office/uml-diagrams-in-visio-ca4e3ae9-d413-4c94-8a7a-38dac30cbed6?ui=en-us&rs=en-us&ad=us

Automatically compiling class diagrams from your source code (useful for acquiring a visual overview of your system in terms of your source code classes):

1. ObjectAid (https://www.objectaid.com/home). Outstanding tool. Read the short instructions how to use within Eclipse.

7.Notes

- **An SRS and an SDD example for a sample project can be found on the Resources/Project section on OWL. You are NOT required to use the same format.**

APPENDIX

Appendix I.A

Example snippet of code for accessing a web site using an http GET request from within a Java program, retrieving a JSON stream, and extracting the information regarding the confirmed Covid-19 cases. You may want to try typing on your browser the URL https://api.covid19api.com/total/dayone/country/italy/status/confirmed to get an idea of the format of the returned data.

```
private double retrieveData(String country) {
	String urlString =
	String.format("https://api.covid19api.com/total/dayone/country/%s/status/confi
	rmed", country);
	System.out.println(urlString);
	int cases = 0;
		try {
			URL url = new URL(urlString);
			HttpURLConnection conn = (HttpURLConnection)url.openConnection();
			conn.setRequestMethod("GET");
			conn.connect();
			int responsecode = conn.getResponseCode();
			if (responsecode == 200) {
			  String inline = "";
			  Scanner sc = new Scanner(url.openStream());
			  while (sc.hasNext()) {
				inline += sc.nextLine();
			  }
			  sc.close();
			  JsonArray jsonArray = new
					JsonParser().parse(inline).getAsJsonArray();
			  int size = jsonArray.size();
			  cases = jsonArray.get(
				   size - 1).getAsJsonObject().get("Cases").getAsInt();
			  System.out.println("Cases: " + cases);
			  }

		} catch (IOException e) {
			  // TODO Auto-generated catch block
			  e.printStackTrace();
		}
		return cases;
	}
```

The java libraries used in the above piece of code are:

```
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;

import com.google.gson.JsonArray;
import com.google.gson.JsonParser;
```

Appendix I.B

Getting the right coordinates in the jpeg given the latitude, the longitude of the country and the size of the jpeg map. See below how to get the jpeg map dimensions.

```java
private Point getXY(double lat, double lng, int mapWidth, int mapHeight) {
	int screenX = (int) Math.round((((lng + 180) / 360) * mapWidth));
	int screenY = (int) Math.round(((((lat * -1) + 90) / 180) * mapHeight));
	return new Point(screenX, screenY);
	}
```

For getting the size of the jpeg map

```java
	myPicture = ImageIO.read(new File("map.jpg"));
	int mapWidth = myPicture.getWidth();
	int mapHeight = myPicture.getHeight();
```

………………………………………………………………

………………………………………………………………

Appendix I.C

Rendering a proportional circle for a country as a function of the maximum value of the analysis results. For example, if Italy had the maximum number of cases between all countries considered with a value of 10,000 then the variable maxValue will be equal to 10,000.

For computing the proportional size of the circle as a function of the max value.

For the sample rendering you see the Figure 7 the values of the above snippet of code where computed as follows (for the Covid Confirmed Cases analysis):

```
int maxOvalDimension;

if (maxValue < 10000)
       maxOvalDimension = 20;
       else if (maxValue < 50000)
              maxOvalDimension = 30;
       else if (maxValue < 100000)
              maxOvalDimension = 50;
       else
              maxOvalDimension = 70;

int minOvalDimension = 15;
```

and

```
int ovalDimension = (int)Math.round(((maxOvalDimension - minOvalDimension) *

                                     percentage) + minOvalDimension);
```

For rendering the circle

```
 Point2D coords = new Point2D.Double(longitude, latitude);
// longitude and latitude values are of type double as given
// from the csv coordinates file


 ………………………………………………………

 ………………………………………………………


 System.out.println("Coordinates: " + coords.getX() + ", " + coords.getY());
 Point testPoint = getXY(coords.getY(), coords.getX(), mapWidth, mapHeight);
 // Add the circle to the image

 Graphics2D editableImage = (Graphics2D) myPicture.getGraphics();
 editableImage.setColor(Color.RED);
 editableImage.setStroke(new BasicStroke(3));
 editableImage.fillOval(testPoint.x - (ovalDimension / 2),
                        testPoint.y - (ovalDimension / 2),
                        ovalDimension,
                        ovalDimension);
```

Appendix I.D

Java libraries used in the sample code for rendering the results

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Point;
import java.awt.geom.Point2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Map;
import java.util.Map.Entry;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
```