Balazs Barcza

SB18002

Cloud Based Web Technologies - Project

Table of Contents

Section 1: The project

Cloud Based Web Technologies - Project
We want to develop a system that delivers the time now in different formats and timezones. This system will based on microservices (do not worry, it is easier than what it sounds) but let's start simple.

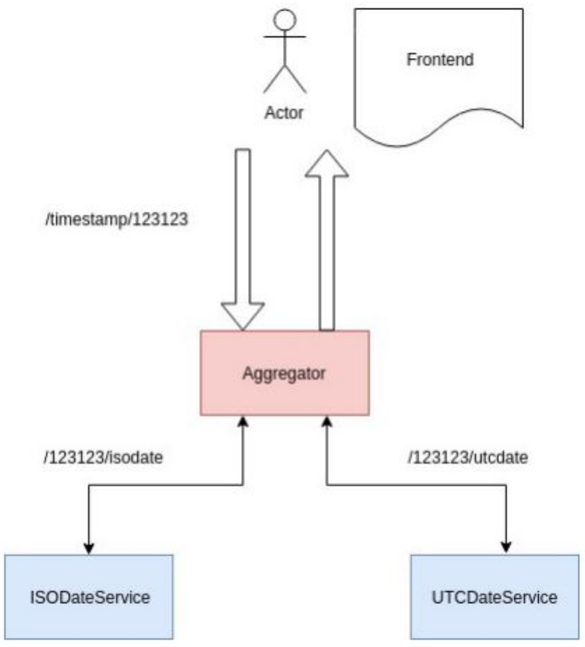
Section 2: Structure

The system is composed of 3 services:
- Aggregator
- Iso date formatter

  I**SO** *Data elements and interchange formats – Information interchange – Representation of dates and times* is an international standard covering the exchange of date- and time-related data. It was issued by the International Organization for Standardization (ISO) and was first published in 1988. The purpose of this standard is to provide an unambiguous and well-defined method of representing dates and times, so as to avoid misinterpretation of numeric representations of dates and times, particularly when data are transferred between countries with different conventions for writing numeric dates and times.
- Utc date formatter

  Coordinated Universal Time (abbreviated to **UTC**) is the primary time standard by which the world regulates clocks and time. It is within about 1 second of mean solar time at 0° longitude,[and is not adjusted for daylight saving time. In some countries where English is spoken, the term Greenwich Mean Time(GMT) is often used as a synonym for UTC and predates UTC by nearly 300 years.

These three components interact according the following schema:



Section 3: Project Description

The service UTCDateService from the above diagram has already been created in class. The ISODateService can be found in the following address:

http://isodate.kubernetes.cafe

```
{"statusCode":404,"error":"Not Found"}
```

As an example, if you browse the URL: http://isodate.kubernetes.cafe/123123123/isodate it should return a timestamp formatted as ISO Date.

```
{"date":"1973-11-26T00:52:03.000Z"}
```

The service UTC Date was built in class and you should be running it in your own machine for the aggregator to be able to consume it.

On this repository: https://github.com/dgonzalez/cct-aggregator You can find the template for working on the aggregator. Here is where we are going to work.

The code, at the moment, is consuming the ISODate service on the address mentioned above. We need to:

Section 4: Project

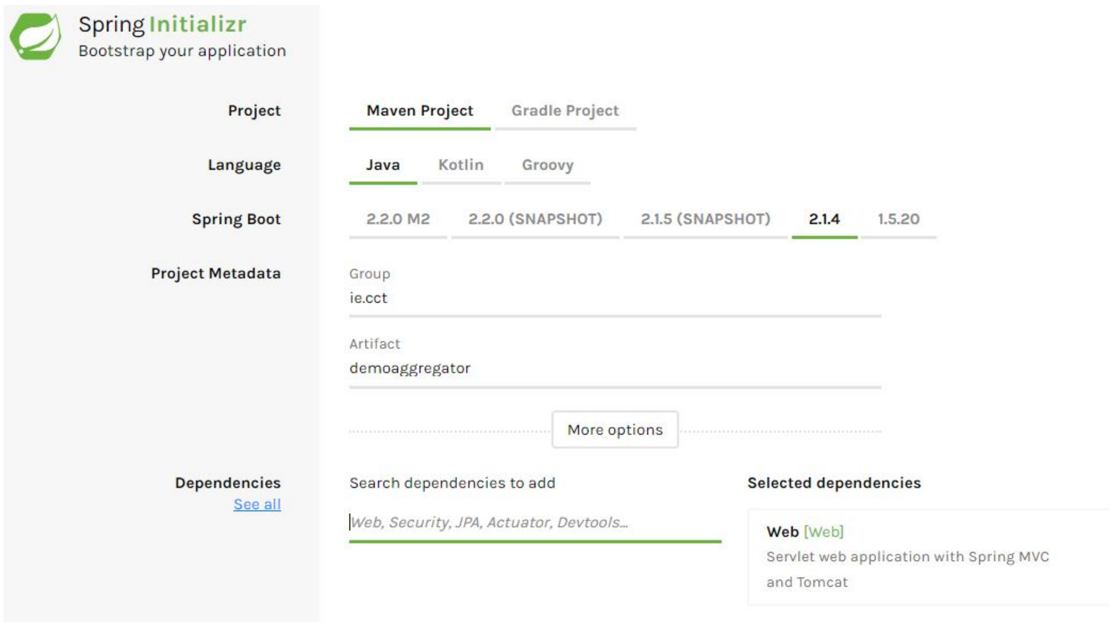
Step 1:

I had to fix the aggregator program because This program missed the Maven files.

Apache Maven:

> Maven is a build automation tool used primarily for Java projects. Maven addresses two aspects of building software: first, it describes how software is built, and second, it describes its dependencies.

I have visited the website and I created a new project.
https://start.spring.io/
I have set up the project and I have to download the zip file.



Spring Initializr
Bootstrap your application

Project	Maven Project [✓] Gradle Project
Language	Java [✓] Kotlin Groovy
Spring Boot	2.2.0 M2 2.2.0 (SNAPSHOT) 2.1.5 (SNAPSHOT) 2.1.4 [✓] 1.5.20
Project Metadata	Group: ie.cct; Artifact: demoaggregator

More options

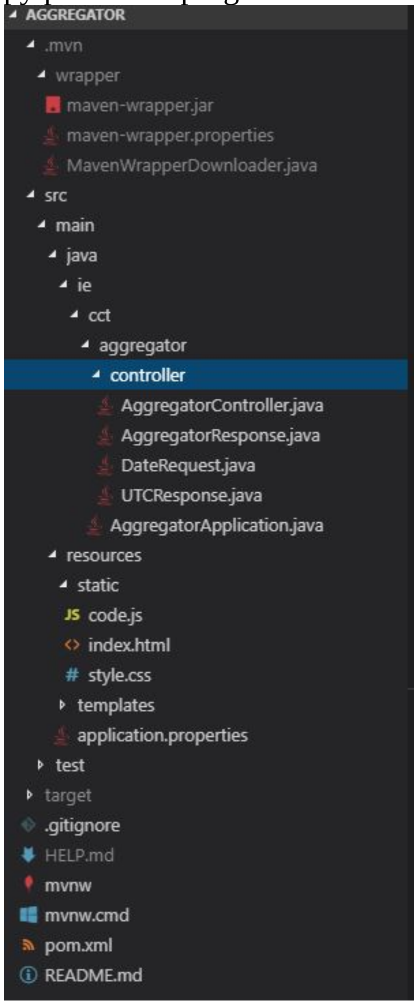
Dependencies
See all

Search dependencies to add
Web, Security, JPA, Actuator, Devtools...

Selected dependencies
Web [Web]
Servlet web application with Spring MVC and Tomcat

https://start.spring.io/

I create all of the files and I have copy pasted the program from the aggregator project.

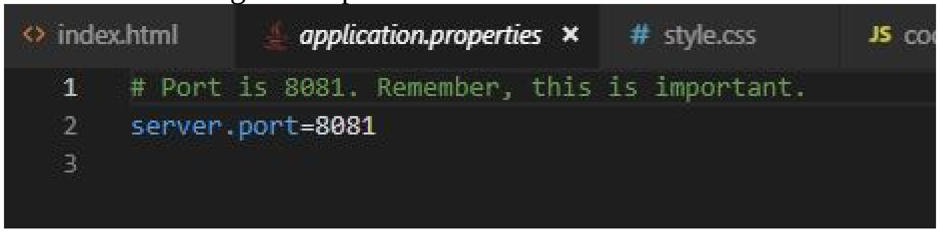


```
AGGREGATOR
  .mvn
    wrapper
      maven-wrapper.jar
      maven-wrapper.properties
      MavenWrapperDownloader.java
  src
    main
      java
        ie
          cct
            aggregator
              controller
                AggregatorController.java
                AggregatorResponse.java
                DateRequest.java
                UTCResponse.java
              AggregatorApplication.java
      resources
        static
          code.js
          index.html
          style.css
        templates
        application.properties
    test
  target
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
  README.md
```

Step 2:

- Consume UTC Date from your local machine and return the UTC and ISO formats together on a single JSON payload. In order to facilitate that, Aggregator is listening on the port 8081 (have a look at the application.properties file) so you can run the aggregator service on the port 8081 and the UTC date service on the port 8080. Remember, to access aggregator, you need to browse http://localhost:8081 (10 Marks)

The aggregator service is listening on the port 8081.



```
# Port is 8081. Remember, this is important.
server.port=8081
```

If We leave empty this file It will use the default port 8080, but we can use any other ports.

Network port:

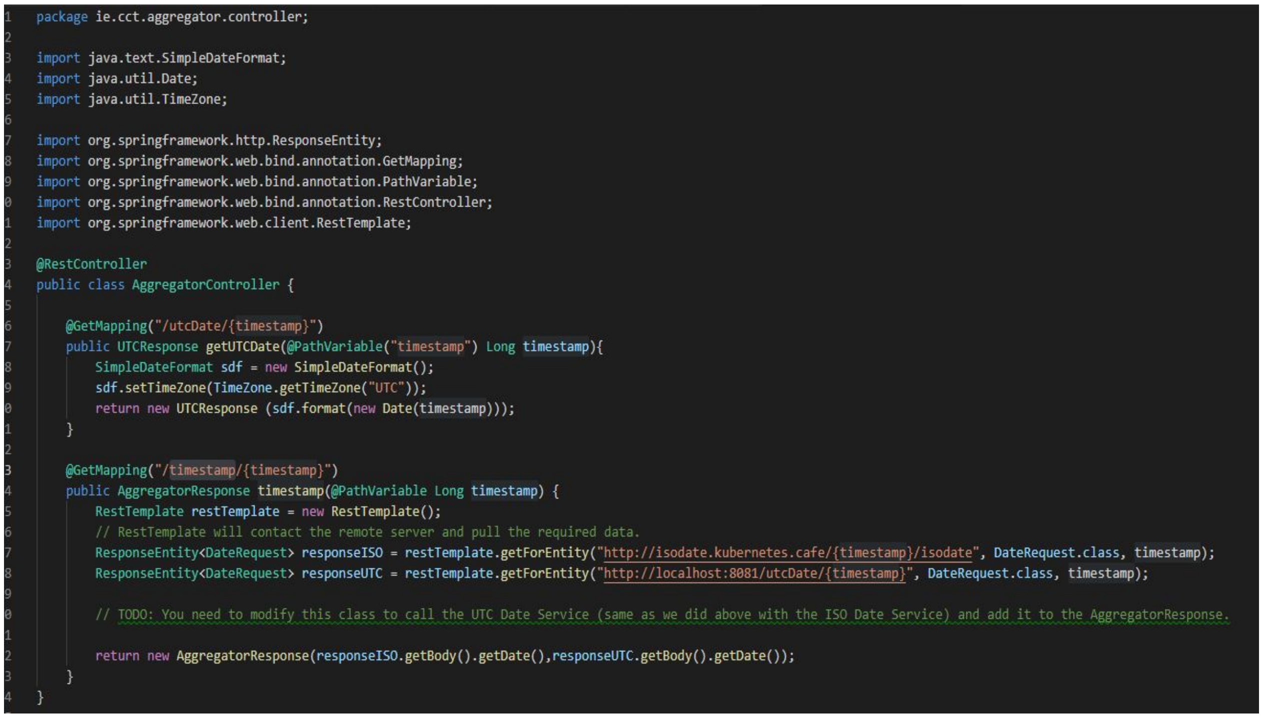
*A **network port** is a number that identifies one side of a connection between two computers. Computers use **port** numbers to determine to which process or application a message should be delivered. As **network** addresses are like street address, **port** numbers are like suite or room numbers.*

Setting up the services:

I have completed up to the UTC service. The UTC is running on the local machine, the ISO is running on the cloud.

Mapping the UTC date service on the local machine

`@GetMapping("/utcDate/{timestamp}")`



```java
package ie.cct.aggregator.controller;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class AggregatorController {

    @GetMapping("/utcDate/{timestamp}")
    public UTCResponse getUTCDate(@PathVariable("timestamp") Long timestamp){
        SimpleDateFormat sdf = new SimpleDateFormat();
        sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
        return new UTCResponse (sdf.format(new Date(timestamp)));
    }

    @GetMapping("/timestamp/{timestamp}")
    public AggregatorResponse timestamp(@PathVariable Long timestamp) {
        RestTemplate restTemplate = new RestTemplate();
        // RestTemplate will contact the remote server and pull the required data.
        ResponseEntity<DateRequest> responseISO = restTemplate.getForEntity("http://isodate.kubernetes.cafe/{timestamp}/isodate", DateRequest.class, timestamp);
        ResponseEntity<DateRequest> responseUTC = restTemplate.getForEntity("http://localhost:8081/utcDate/{timestamp}", DateRequest.class, timestamp);

        // TODO: You need to modify this class to call the UTC Date Service (same as we did above with the ISO Date Service) and add it to the AggregatorResponse.

        return new AggregatorResponse(responseISO.getBody().getDate(),responseUTC.getBody().getDate());
    }
}
```

```
public UTCResponse getUTCDate(@PathVariable("timestamp") Long timestamp){
SimpleDateFormat sdf = new SimpleDateFormat();
sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
return new UTCResponse (sdf.format(new Date(timestamp)));
}
```

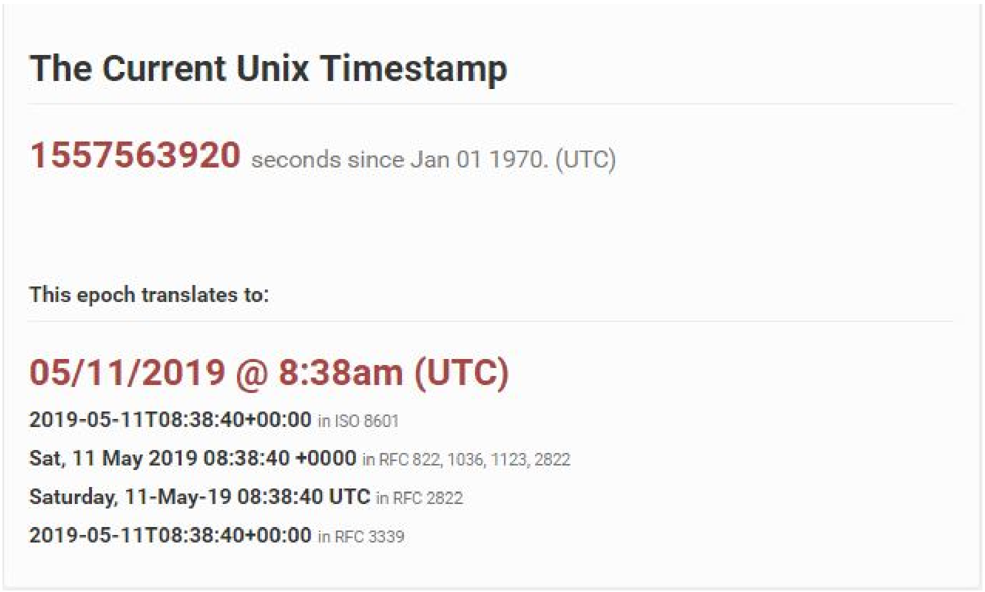
mapping: /*utcDate*/{insert the timestamp}

I had to import java utility.

```
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
```

timestamp:

> *A **timestamp** is a sequence of characters or encoded information identifying when a certain event occurred, usually giving date and time of day, sometimes accurate to a small fraction of a second.*



The Current Unix Timestamp

1557563920 seconds since Jan 01 1970. (UTC)

This epoch translates to:

05/11/2019 @ 8:38am (UTC)

2019-05-11T08:38:40+00:00 in ISO 8601

Sat, 11 May 2019 08:38:40 +0000 in RFC 822, 1036, 1123, 2822

Saturday, 11-May-19 08:38:40 UTC in RFC 2822

2019-05-11T08:38:40+00:00 in RFC 3339

Timestamp is long value.

> ***Java**.Lang.**Long** class in **Java**. **Long** class is a wrapper class for the primitive type **long** which contains several methods to effectively deal with a **long** value like converting it to a string representation, and vice-versa. An object of **Long** class can hold a single **long** value.*

This is java code to conventions timestamp to Utc date:

```
SimpleDateFormat sdf = new SimpleDateFormat();
sdf.setTimeZone(TimeZone.getTimeZone("UTC"));
return new UTCResponse (sdf.format(new Date(timestamp)));
```

The RestTemplate is contacted the remote server and pull the required data.

ResponseISO connection with the cloud.

```
ResponseEntity<DateRequest> responseISO =
restTemplate.getForEntity("http://isodate.kubernetes.cafe/{timestamp}/isodate",
DateRequest.class, timestamp);
```

ResponseUTC connection with the localhost.

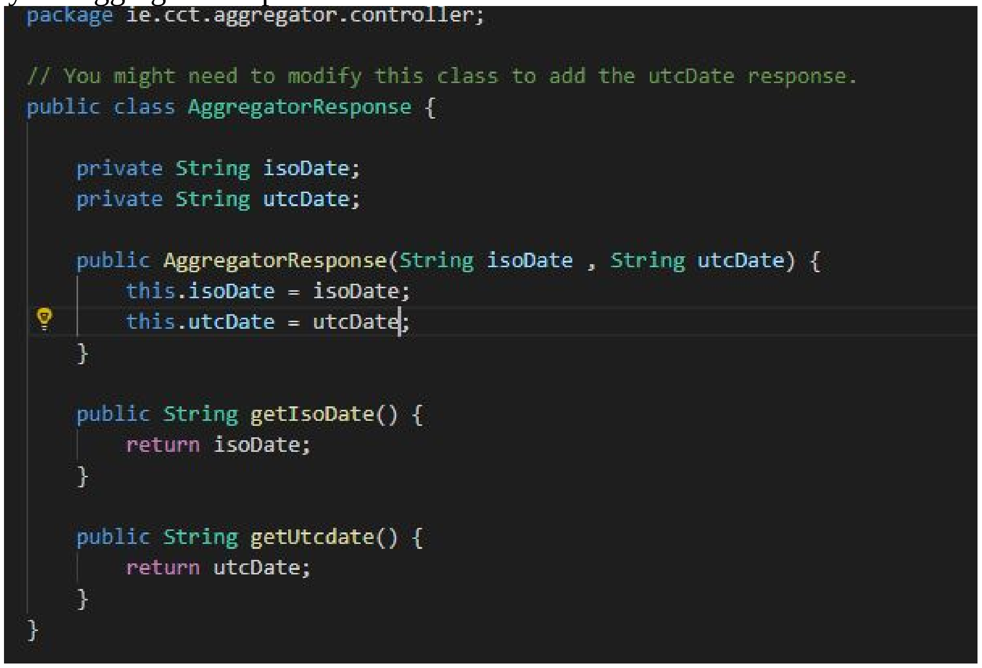
```
ResponseEntity<DateRequest> responseUTC =
restTemplate.getForEntity("http://localhost:8081/utcDate/{timestamp}",
DateRequest.class, timestamp);
```

I have modified the return

```
return new
AggregatorResponse(responseISO.getBody().getDate(),responseUTC.getBody().getDate());
```

Step 3:

I had to modify the aggregator response file.



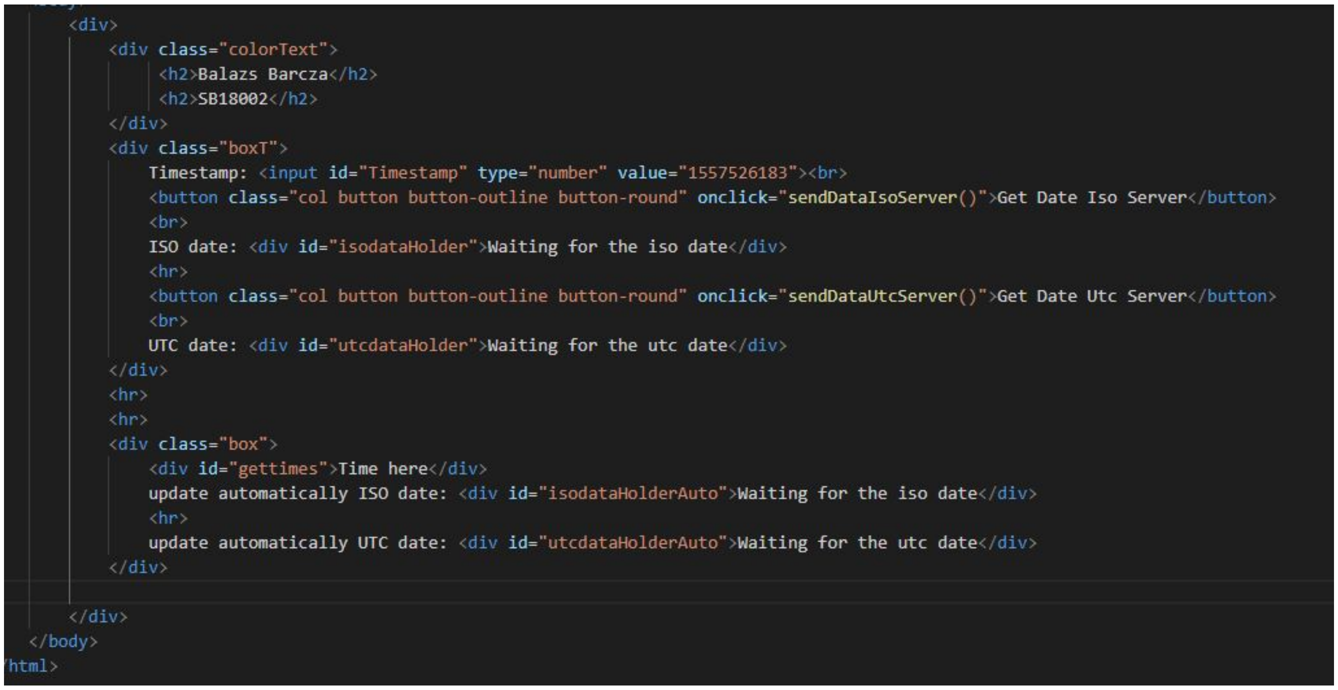
```
package ie.cct.aggregator.controller;

// You might need to modify this class to add the utcDate response.
public class AggregatorResponse {

    private String isoDate;
    private String utcDate;

    public AggregatorResponse(String isoDate , String utcDate) {
        this.isoDate = isoDate;
        this.utcDate = utcDate;
    }

    public String getIsoDate() {
        return isoDate;
    }

    public String getUtcdate() {
        return utcDate;
    }
}
```

Added utcDate value and passed a string “utcDate” to the constructor.
And added getter to the utcdate

Step 4:

I have created index.html file

- input tag “type = number”
- place holder <div> tag
- buttons to get data I have using function
- time place holder where is automation update

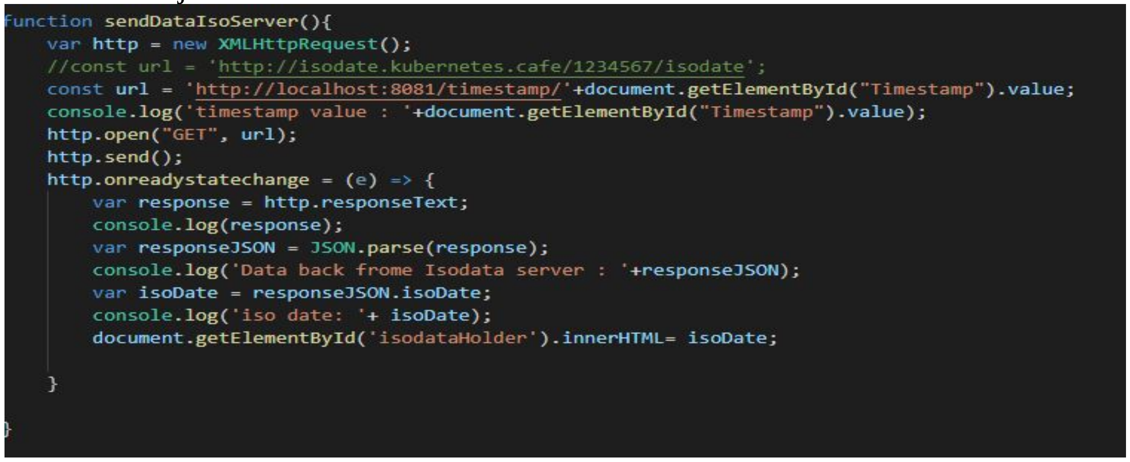


```
    <div>
        <div class="colorText">
            <h2>Balazs Barcza</h2>
            <h2>SB18002</h2>
        </div>
        <div class="boxT">
            Timestamp: <input id="Timestamp" type="number" value="1557526183"><br>
            <button class="col button button-outline button-round" onclick="sendDataIsoServer()">Get Date Iso Server</button>
            <br>
            ISO date: <div id="isodataHolder">Waiting for the iso date</div>
            <hr>
            <button class="col button button-outline button-round" onclick="sendDataUtcServer()">Get Date Utc Server</button>
            <br>
            UTC date: <div id="utcdataHolder">Waiting for the utc date</div>
        </div>
        <hr>
        <hr>
        <div class="box">
            <div id="gettimes">Time here</div>
            update automatically ISO date: <div id="isodataHolderAuto">Waiting for the iso date</div>
            <hr>
            update automatically UTC date: <div id="utcdataHolderAuto">Waiting for the utc date</div>
        </div>

    </div>
  </body>
/html>
```

Step 5:

- Build a UI that allows the user to request a given timestamp to be formatted as UTC and ISO with a call to the Aggregator (10 Marks)

I have created a code.js file



```
function sendDataIsoServer(){
    var http = new XMLHttpRequest();
    //const url = 'http://isodate.kubernetes.cafe/1234567/isodate';
    const url = 'http://localhost:8081/timestamp/'+document.getElementById("Timestamp").value;
    console.log('timestamp value : '+document.getElementById("Timestamp").value);
    http.open("GET", url);
    http.send();
    http.onreadystatechange = (e) => {
        var response = http.responseText;
        console.log(response);
        var responseJSON = JSON.parse(response);
        console.log('Data back frome Isodata server : '+responseJSON);
        var isoDate = responseJSON.isoDate;
        console.log('iso date: '+ isoDate);
        document.getElementById('isodataHolder').innerHTML= isoDate;

    }

}
```

This function is getting iso date from the google cloud.

A common use of JSON is to exchange data to/from a web server.

When receiving data from a web server, the data is always a string.

Parse the data with JSON.parse(), and the data becomes a JavaScript object.

We received this text from a web server:

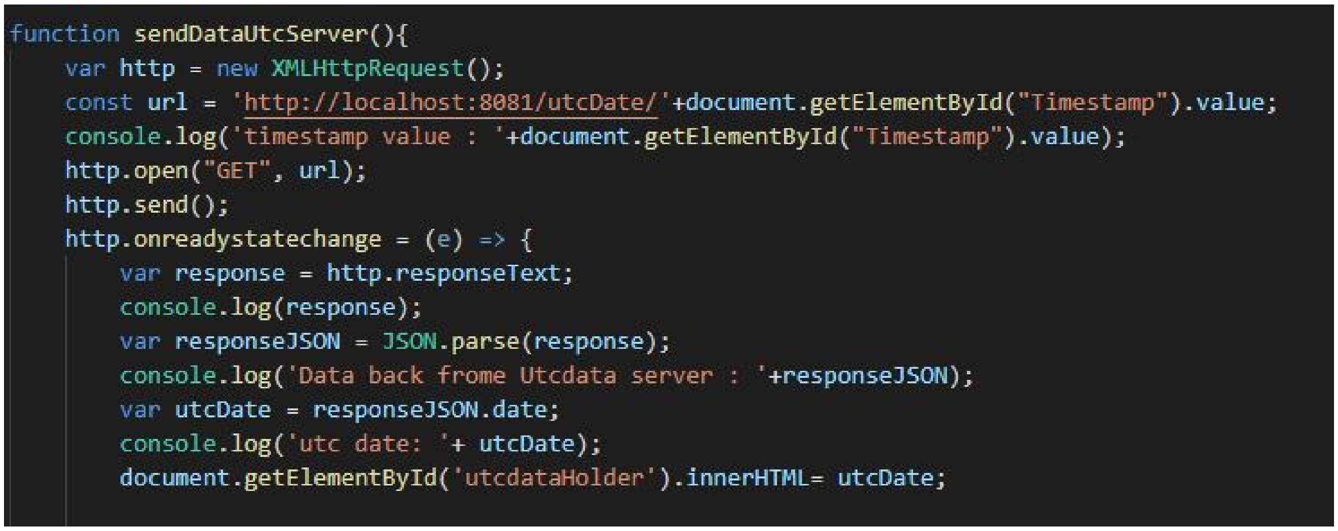
{"isoDate":"1970-01-15T06:15:23.000Z","utcdate":null}

Use the JavaScript function JSON.parse() to convert text into a JavaScript object:

var responseJSON = Json.parse(response)

var isoDate = responseJson.isoDate

1970-01-15T06:15:23.000Z



```
function sendDataUtcServer(){
    var http = new XMLHttpRequest();
    const url = 'http://localhost:8081/utcDate/'+document.getElementById("Timestamp").value;
    console.log('timestamp value : '+document.getElementById("Timestamp").value);
    http.open("GET", url);
    http.send();
    http.onreadystatechange = (e) => {
        var response = http.responseText;
        console.log(response);
        var responseJSON = JSON.parse(response);
        console.log('Data back frome Utcdata server : '+responseJSON);
        var utcDate = responseJSON.date;
        console.log('utc date: '+ utcDate);
        document.getElementById('utcdataHolder').innerHTML= utcDate;
```

This function is getting UTC date from the localhost.

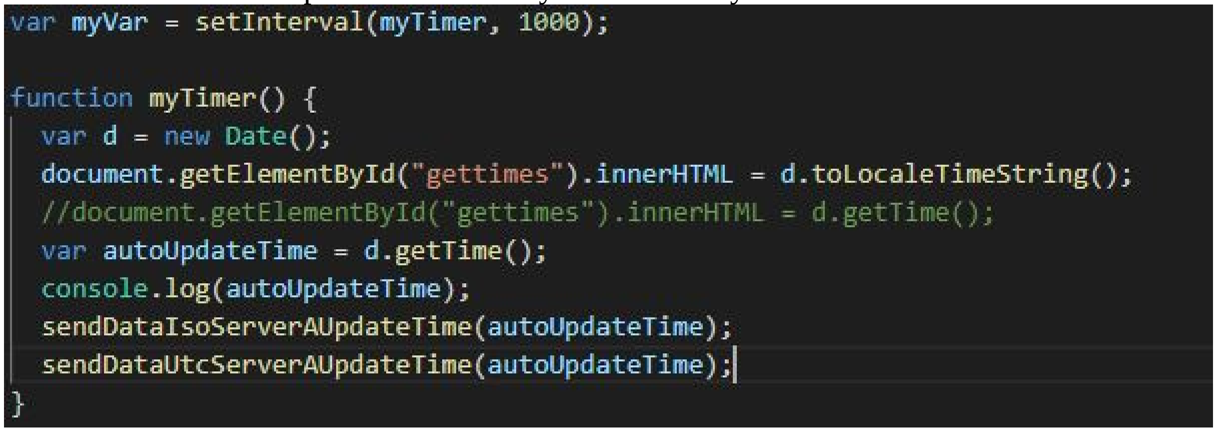
http://localhost:8081/utcDate/ + document.getElementById("Timestamp").value;
localhost + port 8081 + mapping utcDate + get element from the user label: Timestamp

Step 6:
- Using jQuery (and HTML if necessary) create a box that will update automatically every second, formatting the current timestamp into ISO and UTC date formats (5 Marks)

- created stlye.css file to get the box

```
.box{
        background-color: lightgrey;
        width: 300px;
        border: 15px solid green;
        padding: 50px;
        margin: 20px;
```

- created function to update automatically the time every 1 second



```
var myVar = setInterval(myTimer, 1000);

function myTimer() {
  var d = new Date();
  document.getElementById("gettimes").innerHTML = d.toLocaleTimeString();
  //document.getElementById("gettimes").innerHTML = d.getTime();
  var autoUpdateTime = d.getTime();
  console.log(autoUpdateTime);
  sendDataIsoServerAUpdateTime(autoUpdateTime);
  sendDataUtcServerAUpdateTime(autoUpdateTime);
}
```

passing variables to the functions to display the ISO and UTC date

Section 5: Questions

1. The components (aggregator, isodate, utcdate) are using HTTP or HTTPS? Explain what is the difference between the two protocols. (2 marks)

This project I am using the HTTP protocol. The better to change to HTTPS because It is saver and give more protection. The https is had highl levels encryption.

What is HTTP?
HTTP is Hypertext transfer protocol. Simply put - Rules to sending and receiving text-based messages. As we all know, computers work in a language of 1's and 0's i.e. Binary language. Therefore, potentially every set of 1's and 0's construct something, it could be a word.

Let's say I want to write 'a'. Now, if 0 stands for 'a', 1 stands for 'b', and 01 stands for 'c', I can infer that a combination of 0's and 1's can construct a word as well. In this case, the text is already constructed and is being sent on the wire. The computer works on many languages - pure binary, text and some other formats like byte codes. Here, what is being transferred is text. I am emphasizing on 'text' because this text is interpreted by the browser and the moment browser interprets it, it becomes hypertext, and the protocol that transfers the text is referred to as hypertext transfer protocol - HTTP.

Using HTTP, you can definitely transfer images and text and even sound, but not videos.

What is HTTPS?
Hyper Text Transfer Protocol Secure (HTTPS) is the secure version of HTTP, the protocol over which data is sent between your browser and the website that you are connected to. The 'S' at the end of HTTPS stands for 'Secure'. It means all communications between your browser and the website are encrypted. HTTPS is often used to protect highly confidential online transactions like online banking and online shopping order forms.

What is the importance of HTTPS?
We agreed upon the fact that what is being transferred from one point to another is text. To understand

why HTTPS protocol, we first should know how wi-fi routers function.Let's say you are at an airport and you are connecting to the wi-fi which is the property of a third party. Now, when you are communicating over HTTP, the text is being transferred by their router. And if I go to a low version of the router, I can comfortably check and read the text that is being transferred. There could be a password that I can use to login to your bank site and do a fraudulent transaction!. Point being - this is fundamentally insecure. This is called the man in the middle attack.

And this why do we need https when HTTP seems to suffice.

Now, to save our data from such attacks, we need to encrypt that data.

2. When you use the jQuery method $.get(…), what happens at the networking level from your browser? (3 marks)

jQuery's get() method is a convenience function that can be used for making a simple GET request.

jQuery get() Method
The jQuery get() method sends asynchronous http GET request to the server and retrieves the data.

$.get(url, [data],[callback]);
Parameters Description:

- url: request url from which you want to retrieve the data
- data: data to be sent to the server with the request as a query string
- callback: function to be executed when request succeeds

The following example shows how to retrieve data from a text file.

```
$.get('/data.txt',  // url
      function (data, textStatus, jqXHR) {  // success callback
          alert('status: ' + textStatus + ', data:' + data);
    });
```

In the above example, first parameter is a url from which we want to retrieve the data. Here, we want to retrieve data from a txt file located at mydomain.com/data.txt. Please note that you don't need to give base address.
The second parameter is a callback function that will be executed when this GET request succeeds. This callback function includes three parameters data, textStatus and jQuery wrapper of XMLHttpRequest object. Data contains response data, textStatus contains status of request and jqXHR is a jQuery XMLHttpRequest object which you can use for further process.

The network layer is the third level of the Open Systems Interconnection Model (OSI Model) and the layer that provides data routing paths for network communication. Data is transferred in the form of packets via logical network paths in an ordered format controlled by the network layer.

Logical connection setup, data forwarding, routing and delivery error reporting are the network layer’s primary responsibilities.
The network layer is considered the backbone of the OSI Model. It selects and manages the best logical path for data transfer between nodes. This layer contains hardware devices such as routers, bridges, firewalls and switches, but it actually creates a logical image of the most efficient communication route and implements it with a physical medium.

Network layer protocols exist in every host or router. The router examines the header fields of all the IP packets that pass through it.

Internet Protocol and Netware IPX/SPX are the most common protocols associated with the network layer.

In the OSI model, the network layer responds to requests from the layer above it (transport layer) and issues requests to the layer below it (data link layer).

Main commands / help:

.\mvnw spring-boot:run	windows running the server
.\mvnw.cmd compile	compile the spring project

email: david.gonzalez@nearform.com