Big Data Application Architecture - Final Project

ywang27@uchicago.edu

This document gives a brief description of my final project - A **Web Application for Querying and Submitting COVID-19 Data**. All the code for the project, inlcuding backend part and frontend part, are submitted along with this document. For simplicity, I select some part of code instead of all the lengthy code to show in this document.

Because I have very limited experience about node.js and web development before, I build my web page based on the fligh-and-weather app. Although I cannot make many improvements to the code to make it rather different from the flight-and-weather app or more beautiful, I have tried my best in this part.

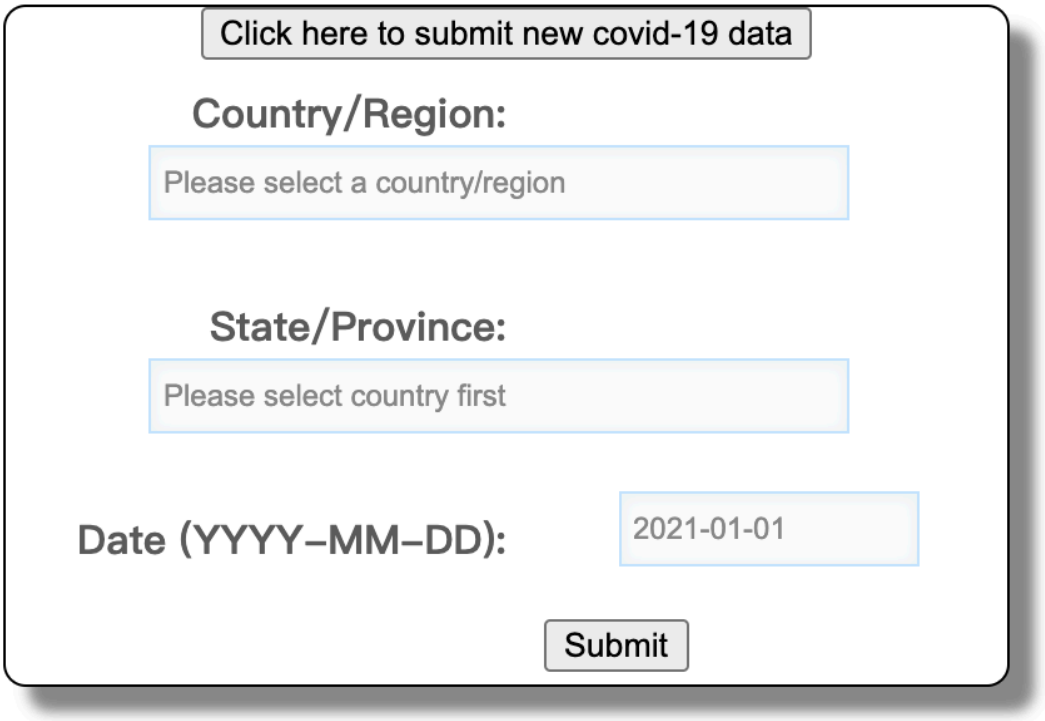
About

The Web Application for Querying and Submitting COVID-19 Data, as its name shown, is designed for express query of global COVID-19 data as well as submitting new covid-19 data occuring at the moment. All the data is collected from COVID-19 Data Repository by the Center for Systems Science and Engineering (CSSE) at Johns Hopkins University. The application consists of two parts.

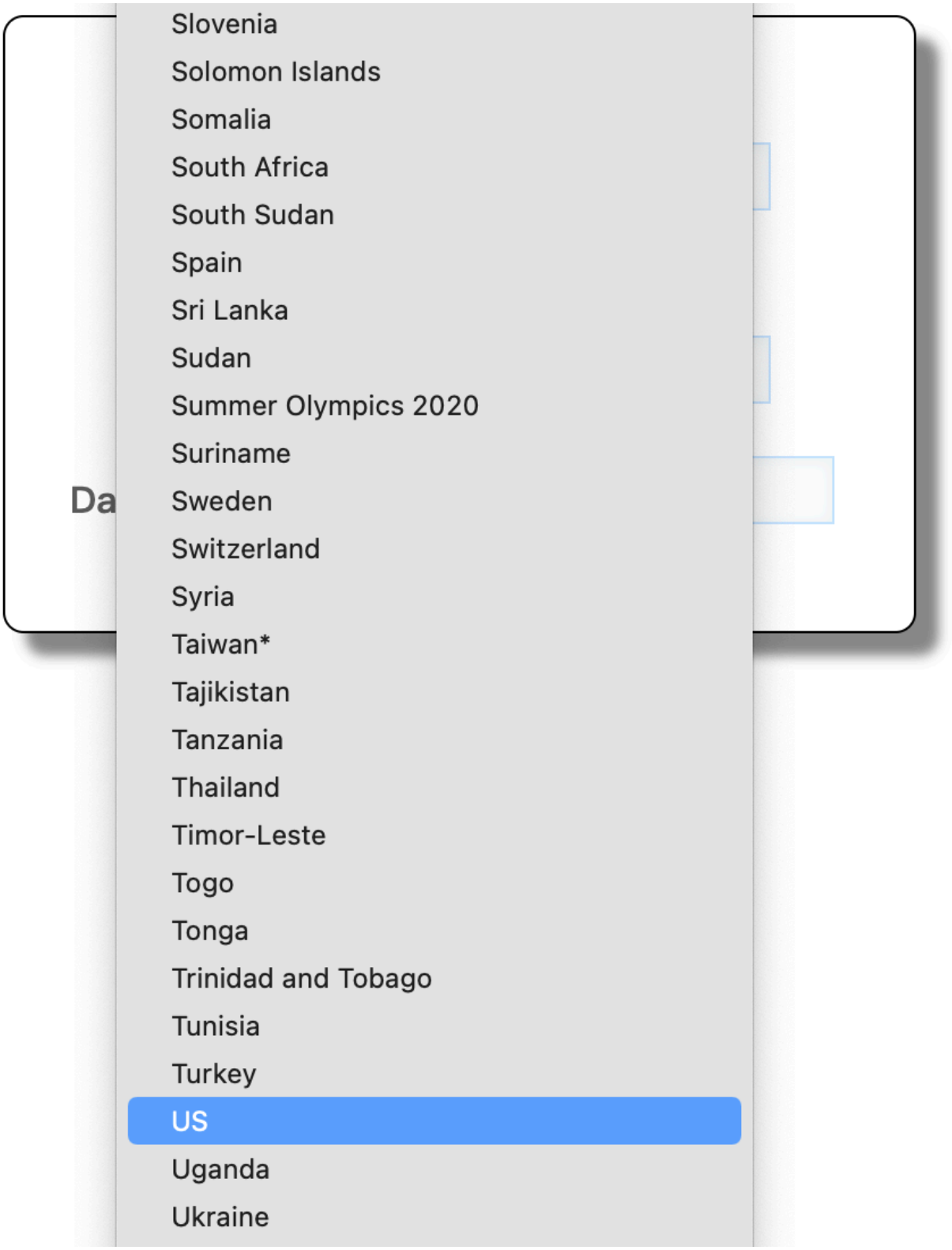
- Qureying data. The application first accepts a country/region the user choosed (for example, United States). Then it shows the states/province of that country/region for further user choice (for example, Illinois). It also accepts a date value (for example, 2021-01-01). The application would show the accumulated data for this location from the beginning date of COVID-19 until the date user enters in, including
    - Confirmed cases
    - Deaths cases
    - Recovered cases
    - Active cases
    - Recovering rate (if applicable)
    - Death rate (if applicable)
- Submitting data. The application allows for submitting new COVID-19 data by providing a interface for user input. Simillarly, a user input a new record by selecting country/region - state/province - date - case statistics. The added record would be stored and could be shown is it is queried.

Below are a couple of screeshots of the running application.

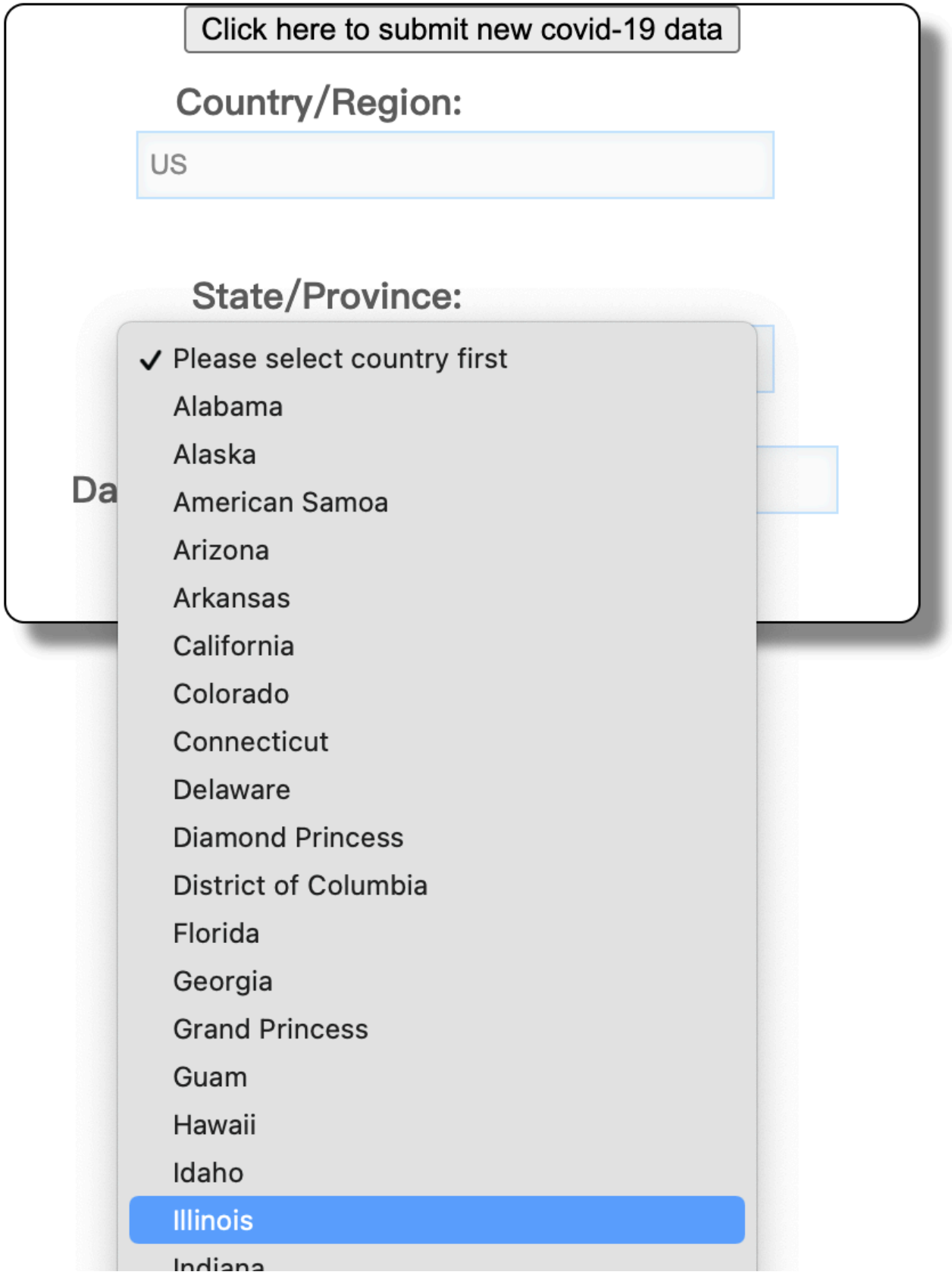
The page for querying the COVID-19 data:



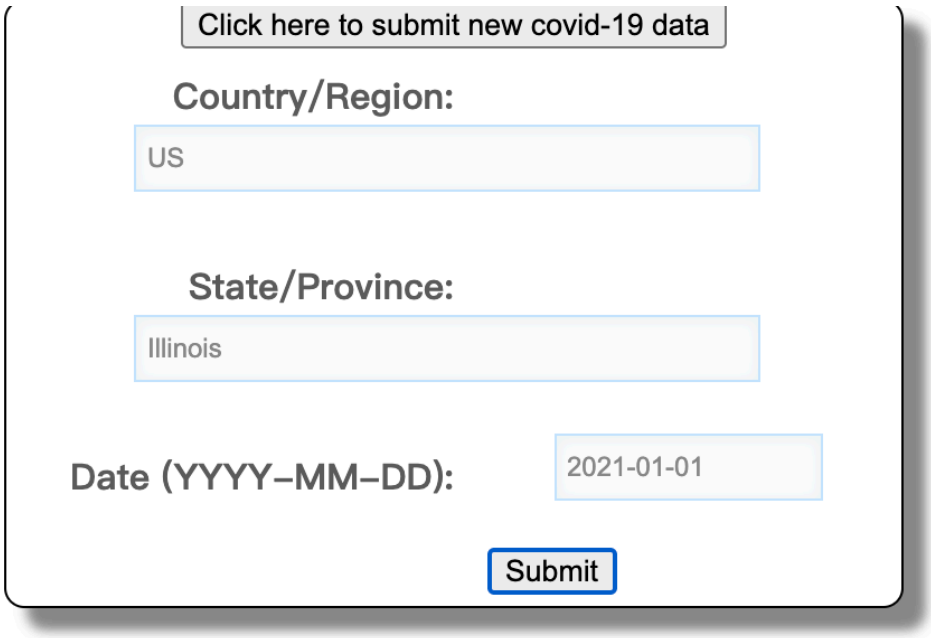
Select a country:



Select a state:



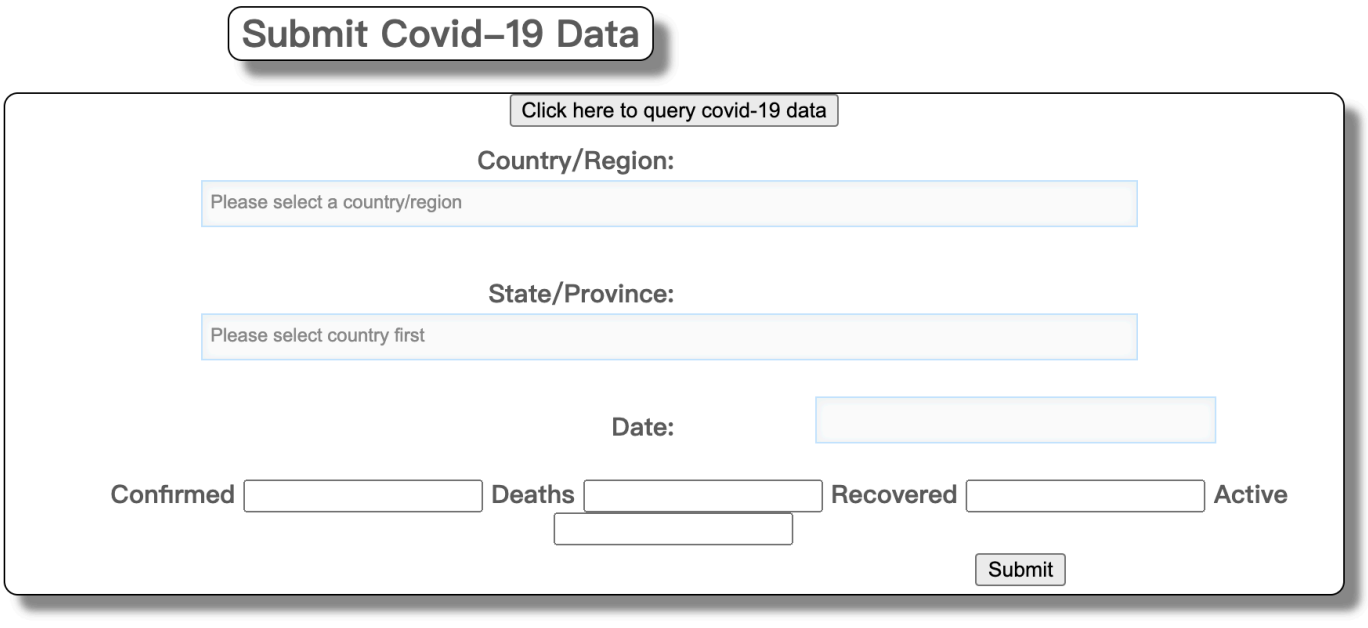
Enter date "2021-01-01" and click submit:



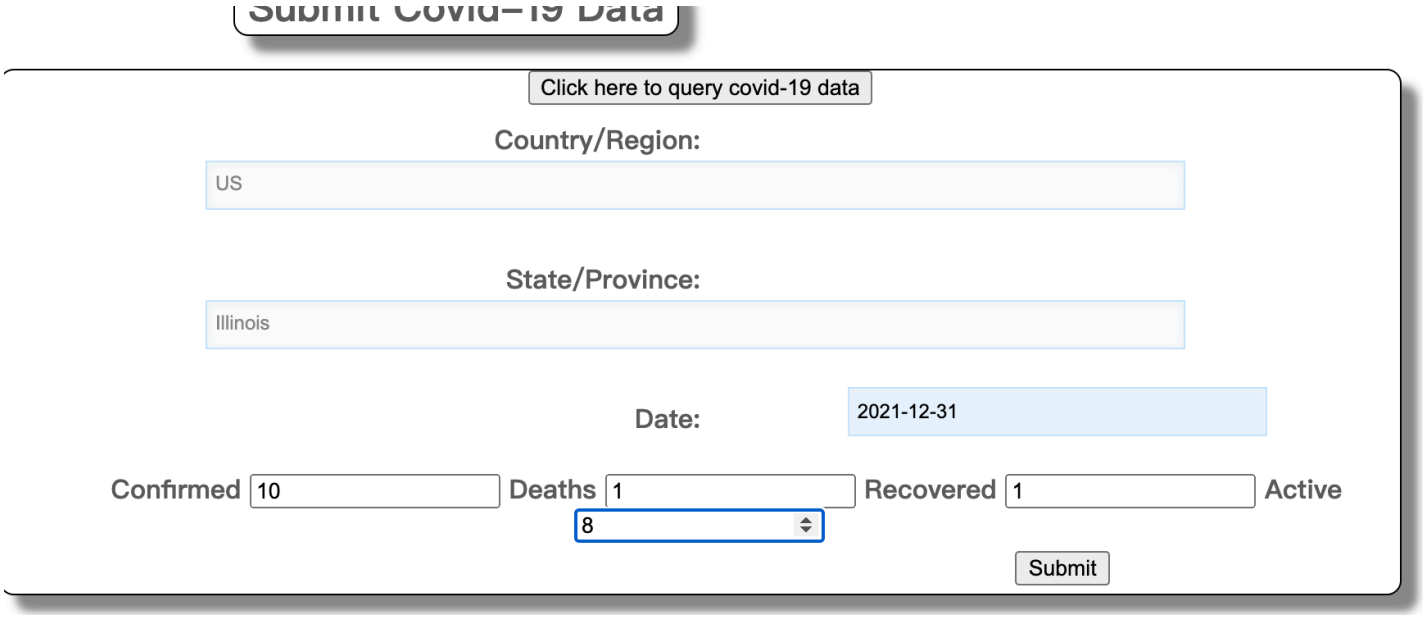
Accumulated Covid-19 Data in US, Illinois until 2021-01-01

Confirmed	Active	Deaths	Recovered	Recovering Rate	Death Rate
963389	945411	17978	0	0%	1.87%

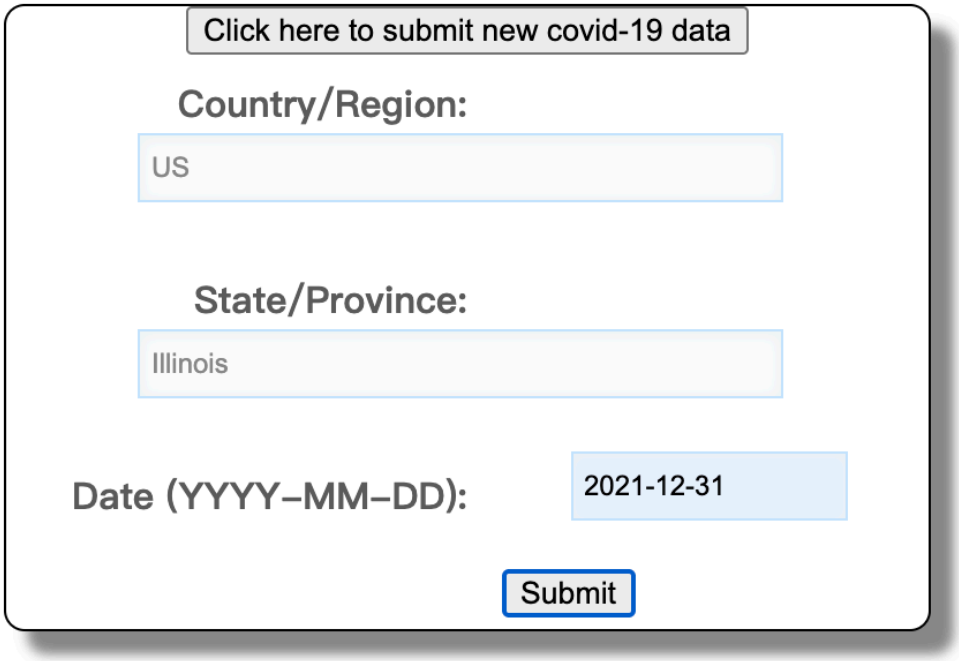
Click "Click here to submit new covid-19 data":



Similiarly, select the country and state, enter the required fields, and click submit:



The return to the query page and query this new record:



Accumulated Covid-19 Data in US, Illinois until 2021-12-31

Confirmed	Active	Deaths	Recovered	Recovering Rate	Death Rate
10	8	1	1	10%	10%

which is the expected outcome.

Commands to Run

All the code are on the cluster `ec2-3-144-71-8.us-east-2.compute.amazonaws.com`, under the path `/home/hadoop/ywang27/final`. The files under `webapp` is also uploaded to `ec2-52-14-115-151.us-east-2.compute.amazonaws.com`, under the `/home/ywang27/final` where the app will be running on.

```
-final
    --BatchLayer
    --ServingLayer
    --SpeedLayer
        ---src
        ---target
    --webapp
        ---src
    --update_datalake_compute_batchview.sh
```

To run the application, first run the speed layer on `ec2-3-144-71-8.us-east-2.compute.amazonaws.com`

```
cd /home/hadoop/ywang27/final/SpeedLayer/target
spark-submit --master local[2] --driver-java-options "-
Dlog4j.configuration=file:///home/hadoop/ss.log4j.properties" --class StreamCovid uber-
SpeedLayer-1.0-SNAPSHOT.jar b-2.mpcs53014-kafka.198nfg.c7.kafka.us-east-
2.amazonaws.com:9092,b-1.mpcs53014-kafka.198nfg.c7.kafka.us-east-2.amazonaws.com:9092
```

The run the node.js on `ec2-52-14-115-151.us-east-2.compute.amazonaws.com` (here I use port 3027)

```
cd /home/ywang27/final/src
node app.js 3027 172.31.39.49 8070 b-1.mpcs53014-kafka.198nfg.c7.kafka.us-east-
2.amazonaws.com:9092,b-2.mpcs53014-kafka.198nfg.c7.kafka.us-east-2.amazonaws.com:9092
```

Go to `http://ec2-52-14-115-151.us-east-2.compute.amazonaws.com:3027/covid-show.html` and the application should be runing.

Where to find ...

- Tables in Hive: all the tables involved in my project are `ywang27_covid_all`, `ywang27_covid`, `ywang27_country_date`, `ywang27_country_date_for_hbase`
- Tables in Hbase: the Hbase table for my project is `ywang27_country_state_date`

Implementation

Datalake - HDFS

First download/update the original dataset from GitHub.

```
git pull
```

The original dataset is a bunch of csv files, eaching containing the data for one day. To create the datalake, I first concat all the cvs files as one file.

```
cat *.csv > all.csv
```

Then load it into **HDFS**

```
hdfs dfs -put all.csv /ywang27
```

Batch Layer - Hive

In **Hive**, create a Hive table `ywang27_covid_all` to store all the data in `all.csv`

```
CREATE EXTERNAL TABLE IF NOT EXISTS `ywang27_covid_all`(
    # fields
)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' NULL DEFINED AS ''
STORED AS TEXTFILE;

LOAD DATA INPATH '/ywang27/all.csv' OVERWRITE INTO TABLE ywang27_covid_all;
```

Filter out the record containing invalid date and save the results to a new table `ywang27_covid`

```
INSERT overwrite TABLE ywang27_covid
SELECT * FROM ywang27_covid_all n
WHERE n.last_update IS NOT NULL;
```

Create a new table `ywang27_country_date` to save batch view, which contains processed `province_state` and `update_date` fields

```
SELECT DISTINCT
    if(province_state is not NULL, province_state, country_region) as province_state,
    to_date(last_update) as update_date,
    # ...
FROM ywang27_covid
```

All the code for [illegible]

All the code for serving layer are in `final/BatchLayer`

Serving Layer - HBase

In **Hbase**, create a table `ywang27_country_state_date` with column family `data`

```
create 'ywang27_country_state_date', 'data'
```

In Hive, create a table `ywang27_country_date_for_hbase`, connect it with `ywang27_country_state_date`, and insert overwrite it by selecting from `ywang27_country_date`

```
CREATE EXTERNAL TABLE IF NOT EXISTS ywang27_country_date_for_hbase (
    # fields
)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler'
WITH SERDEPROPERTIES ('hbase.columns.mapping' =
    ':key,data:confirmed,data:deaths,data:recovered,data:active')
TBLPROPERTIES ('hbase.table.name' = 'ywang27_country_state_date');

INSERT OVERWRITE TABLE ywang27_country_date_for_hbase
SELECT CONCAT(country_region, province_state, update_date),
       # other fields
FROM ywang27_country_date;
```

All the code for serving layer are in `final/ServingLayer`.

The procedure for updating the datalake and batch views could be done by simply running `final/update_datalake_compute_batchview.sh`.

Speed Layer - Spark, Scala, Kafka

First create a topic `ywang27_covid` in **Kafka** for this project

```
kafka-topics.sh --create --zookeeper z-3.mpcs53014-kafka.198nfg.c7.kafka.us-east-2.amazonaws.com:2181,z-2.mpcs53014-kafka.198nfg.c7.kafka.us-east-2.amazonaws.com:2181,z-1.mpcs53014-kafka.198nfg.c7.kafka.us-east-2.amazonaws.com:2181 --replication-factor 1 --partitions 1 --topic ywang27_covid
```

Then create a **Scala** class `KafkaCovidRecord` for messages in Kafka in `KafkaCovidRecord.scala`

```
case class KafkaCovidRecord(
  country: String,
  state: String,
  date: String,
  confirmed: String,
  deaths: String,
  recovered: String,
  active: String
)
```

Then create the object `StreamCovid` in `StreamCovid.scala`.

In this object, create the hbase connection and get connected with `ywang27_country_state_year`

```
val hbaseConnection = ConnectionFactory.createConnection(hbaseConf)
val covid_data = hbaseConnection.getTable(TableName.valueOf("ywang27_country_state_date"))
```

Declare the **SparkStreaming** context with interval 5 seconds

```
val ssc = new StreamingContext(sparkConf, Seconds(5))
```

Set Kafka topic to `ywang27_covid`

```
val topicsSet = Set("ywang27_covid")
```

After we get the serialized records in kafka, read its value using the `KafkaCovidRecord`

```
val kfrs = serializedRecords.map(rec => mapper.readValue(rec, classOf[KafkaCovidRecord]))
```

Finally add the value to Hbase via `org.apache.hadoop.hbase.client.Put`

```
val processed_covid = kfrs.map(addToHbase)

def addToHbase(kfr: KafkaCovidRecord): String = {
    val rowkey = kfr.country + kfr.state + kfr.date
    val put = new Put(Bytes.toBytes(rowkey))
    put.addColumn(Bytes.toBytes("data"), Bytes.toBytes("confirmed"),
Bytes.toBytes(kfr.confirmed))
    put.addColumn(Bytes.toBytes("data"), Bytes.toBytes("active"),
Bytes.toBytes(kfr.active))
    put.addColumn(Bytes.toBytes("data"), Bytes.toBytes("deaths"),
Bytes.toBytes(kfr.deaths))
    put.addColumn(Bytes.toBytes("data"), Bytes.toBytes("recovered"),
Bytes.toBytes(kfr.recovered))
    covid_data.put(put)
    return "Updated speed layer"
}
```

Web - Node.js

The frontend part is quite similiar to flight-and-weather app.

In `app.js`, set the kafka topic to send to as `ywang27_covid`

```
kafkaProducer.send([{ topic: 'ywang27_covid', messages: JSON.stringify(report)}],
    function (err, data) {
      console.log(report);
      res.redirect('covid-submit.html');
    });
```

In order to implement that the second selection option lists (state list) changes as the first selection option list (country list), in function `setstate()` get the id of the second select tag, concatenate the all states/province according to the country value, and assign the cancatenated string to the second select tag - innerHTML property. The function is called when the first selection (country) has been made.

```
function setstate(){
  let obj = document.getElementById("country");
  let country = obj.options[obj.selectedIndex].value;
  let states = country_state[country];
  let numofstates = states.length;
  let output = '<option value="">Please select country first</option>'

  for (var i = 0; i < numofstates; i++){
    output += `<option value="${states[i]}">${states[i]}</option>`
  }

  let $id_state = document.getElementById("state")
```

```
  let $id_state = document.getElementById( 'state' )
  $id_state.innerHTML = output
}
```