



UNIVERSIDAD NACIONAL AUTÓNOMA
DE MÉXICO

FACULTAD DE CIENCIAS

Documentacion

Modelado y Programacion

**Victor Federico Torres Trejo
Diego Castro Rendon**

September 18, 2022

Contents

1	Definicion del problema	2
1.1	Entender el problema	2
1.2	Arsenal	3
1.3	Requisitos funcionales	3
1.4	Requisitos no funcionales	3
2	Analisis del problema	3
3	Seleccion de la mejor alternativa	5
4	Pseudocodigo	6
5	Mantenimiento	7
6	Pruebas	7
6.1	Prueba con el archivo csv	8
6.2	Prueba una ciudad inexistente	10
6.3	Una API erronea	11
7	Costos	12

1 Definicion del problema

Dia a dia miles de personas van y vienen de aeropuertos a aeropuertos, cambiando drasticamente de zona horaria y clima. El aeropuerto de la Ciudad de Mexico te contrata para una tarea, la cual es entregar el informe del clima de la ciudad de salida y la ciudad de llegada para 3 mil tickets que salen el mismo dia que se corre el algoritmo. El programa no busca que sea interactivo, ya que el mercado a quien va dirigido son sobrecargos y pilotos que desconocen de programacion, por lo que solo nos interesara el clima.

1.1 Entender el problema

Se quiere obtener el clima de una ciudad, dado su longitud y latitud mediante el uso de servicios web (API), son suficientes siempre y cuando se tenga una llave, ya que para hacer una request a la API, solo se necesita estos dos datos y una llave valida para permitir obtener datos del clima de cualquier localizacion. Al hacer la request la API, nos devuelve una serie de datos en formato JSON, la cual contiene toda la informacion climatologica de la ubicacion solicitada, entre los cuales esta el clima y temperatura, los cuales son la solucion al problema planteado. Para llegar a esta solucion solo se tiene que especificar longitud, latitud y la llave para que con estos datos se haga una request y nos devuelva los datos en formato JSON, posteriormente hay que procesar estos datos para hacerlos mas legibles y filtrar solo la informacion que nos interesa.

1.2 Arsenal

- Paradigma: orientado a objetos
- Lenguaje: Python
- Herramientas: API, JSON

1.3 Requisitos funcionales

Obtener el clima de la ciudad a donde se va a viajar

1.4 Requisitos no funcionales

Garantizar la eficacia, seguridad al procesar los datos y la resistencia a fallos del programa. El programa debe ser amigable para cualquier persona ajena a los conceptos de computacion y ajena al manejo de una terminal

2 Analisis del problema

Tenemos como entrada un archivo csv con los datos de cada vuelo que contiene el nombre del aeropuerto en version codigo de la ciudad, longitud y latitud, y como entrada del usuario que seleccione la ciudad a la cual va ir. Como salida tenemos datos json que son convertidos a strings legibles y sintetizadas para una mejor comprension El modelo de datos es principalmente a traves de matrices que funcionan como tablas hash donde en cada casilla almacenan la informacion requerida, estas determinan el desempeño en una complejidad de acceso constante aunque ocupan gran espacio, es por eso que tenemos que ver que tamaño nos conviene en relacion tamaño-colisiones donde se busca minimizar ambos, se ajusta a los requisitos funcionales porque al final guarda informacion a la cual posteriormente vamos a acceder como ya se menciono en un costo constante y nos da los datos que requerimos para el siguiente paso o mostrar la informacion dependiendo lo que se requiere. No puede haber algo mejor ya que el acceso es constante y si puede haber algo peor como lo son las listas, pilas, colas o incluso un arreglo donde la informacion almacenada no tiene ninguna relacion con el indice donde esta guardada, son peores porque la busqueda dentro de estas estructuras es de tiempo lineal ya que hay que buscar uno a uno.

3 Seleccion de la mejor alternativa

Para el diseño de la interfaz y minimizar la informacion que tiene que ingresar el usuario se va a filtrar todos los vuelos que hay, dejando seleccionar al usuario la ciudad de origen y una vez hecho esto, se filtran los destinos dejandolo seleccionar unicamente los destinos para los cuales hay un vuelo desde la ciudad seleccionada, ademas que de esta forma evitamos el ingreso de datos erroneos, ya que al ser una seleccion, nosotros controlamos la entrada. Una vez que el lo selecciona armamos una tabla hash de ciudades donde cada elemento de la tabla contiene un arreglo con la ciudad misma y sus destinos posibles, sin repeticiones. Una vez hecho esto se escribe en el archivo json de cada ciudad correspondiente la informacion para su posterior consulta, ya que cuando se selecciona un destino de esta manera filtramos los datos.

En cuanto al manejo de peticiones, sabemos que no podemos hacer mas de 60 por minuto e incluso podemos no hacer algunas ya que si se quiere saber, la informacion de una request realizada hace 10 minutos, pues el clima no ha variado significativamente, por lo que usando tablas hash guardamos la informacion en un diccionario y establecemos condiciones para hacer una request, estas son si ha pasado mas de media hora o si no hay informacion guardada. Una vez que se obtuvo los datos en formato json del clima, se procesan y filtran a datos legibles y se muestran en pantalla. Por ultimo se guarda en un archivo el historial de vuelos realizados considerando que al momento de la consulta es un vuelo que se ha realizado por si despues solo se quiere consultar el vuelo y no registrarlo.

Como vemos el boceto del diseño, el usuario solo puede ingresar 3 cosas al sistema, su llave API, su seleccion de origen(la cual esta acotada por los destinos posibles) y la seleccion de destino se puede una vez que ya se selecciono el origen, filtra de acuerdo a la ciudad de origen los unicos destinos posibles que hay, por lo que a menos que sea la api o un destino vacio lo cual se manejo con excepciones, no se rompe el codigo ni dejamos a su suerte al usuario.

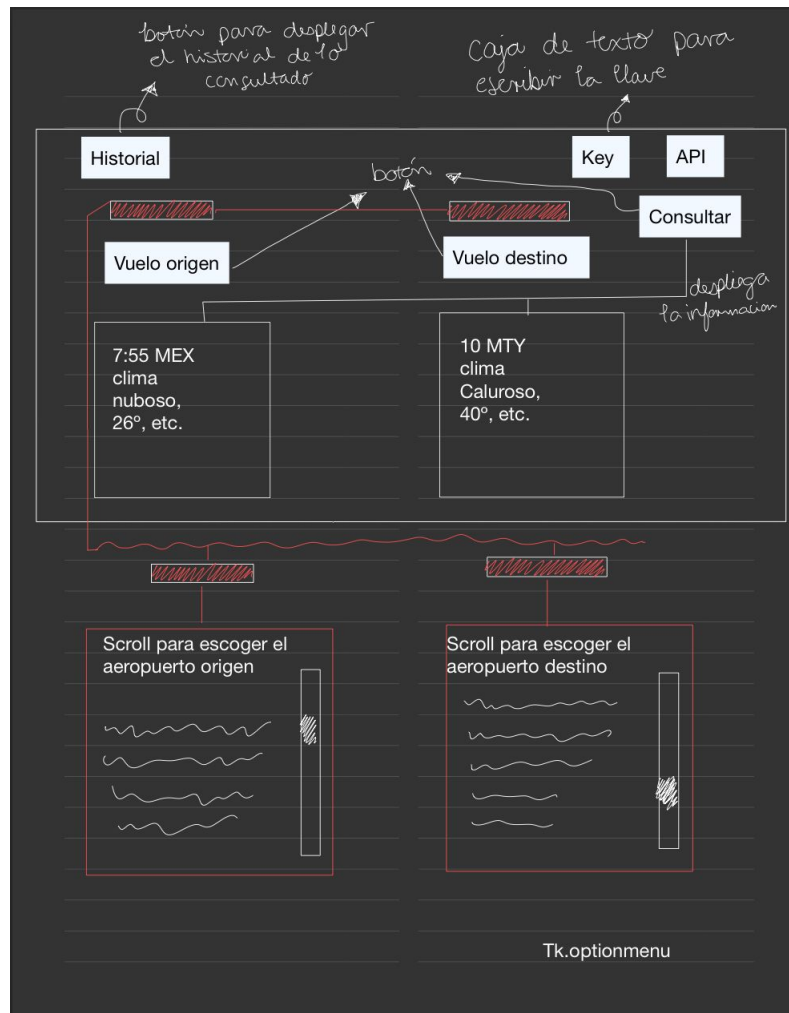


Figure 1:

4 Pseudocodigo

- 1: **for** cada vuelo **do**
- 2: **if** ciudad de origen no esta registrada **then**
- 3: regístrala en la primera casilla vacía
- 4: **end if**
- 5: **if** ciudad de destino no esta registrada **then**
- 6: regístrala en el arreglo de la ciudad de origen

```

7:   end if
8: end for
9: for ciudad de Origen do ciudades
10:   Escribir en cada json el arreglo de ciudad Origen
11: end for
12: Leer ciudad Origen
13: Mostrar todos los destinos posibles dada la ciudad de origen
14: climaOrigen = obtenerClima(ciudadOrigen, api)
15: Leer ciudad destino
16: climaDestino = obtenerClima(ciudadOrigen, api)
17: Mostrar Clima
18: Registrar el vuelo en un archivo
19: function OBTENERCLIMA(ciudadOrigen, api)
20:   if clima ya fue registrado en cache then
21:     if Han pasado mas de 30 min desde la ultima request then
22:       realiza una peticion y guarda en cache
23:     end if
24:   else
25:     realiza la peticion
26:     Guarda en los datos en la primera posicion disponible
27:   end if
28: end function

```

5 Mantenimiento

En cuanto a mantenimiento se refiere si se podria dar seria tener la base de datos actualizada, si cambia el sistema operativo descargar las librerias y lenguajes ademas de que conforme se vaya mejorando la parte de encapsulacion creo que el programa podria acortarse el tamaño del programa.

6 Pruebas

Las pruebas las realizamos ingresando los datos manualmente y desde terminal para poder manipular mejor los datos, ya que como la interfaz es amigable con el usuario, son pocos los datos que el mismo manipula. Identificamos los datos como

6.1 Prueba con el archivo csv

¿Que sucede si ingresamos el archivo csv incorrecto, es decir, nosotros requerimos que el archivo csv venga de la siguiente manera: $x, y, lat_o, lon_o, lat_d, long_d$, donde:

- x : nombre codigo de la ciudad de origen
- y : nombre codigo de la ciudad de destino
- lat_o : latitud de la ciudad de origen
- $long_o$: longitud de la ciudad de origen
- lat_d : latitud de la ciudad de destino
- $long_d$: longitud de la ciudad de destino

por cada fila.

Si ingresamos un csv incorrecto nos mandara un mensaje diciendo que el archivo CSV es incorrecto y terminara la ejecucion del programa:

```
1 victort: ~/.../Proyecto1/WeatherProject git:( victor U:2 ? :1 × )
2   → ^ ls data/
3 dataset1.csv  datos.csv  historial.txt  nVuelo.txt
4 victort: ~/.../Proyecto1/WeatherProject git:( victor U:2 ? :1 × )
5   → ^ python src/prueba.py
6 Formato del CSV incorrecto
7 victort: ~/.../Proyecto1/WeatherProject git:( victor U:2 ? :1 × )
8   → ^ ls data/
9 dataset1.csv  datos.csv  historial.txt  nVuelo.txt
10 victort: ~/.../Proyecto1/WeatherProject git:( victor U:2 ? :1 × )
11  → ^
```

Figure 2: Ingresando un archivo csv incorrecto

Ingresando un CSV bajo el formato correcto, ademas del programa ejecutarse con normalidad, observamos que en la carpeta data se crea unos archivos json por cada ciudad de origen:

```

Actividades  Emacs  10 de sep 14:36
~term - GNU Emacs at victortorres

victort: ~/.../Proyecto1/WeatherProject git:( victor 0/2 7/1 x )
+ ls data/
dataset1.csv  datos.csv  historial.txt  nuevo.txt
victort: ~/.../Proyecto1/WeatherProject git:( victor 0/2 7/1 x )
+ python src/prueba.py
ME
["MTV", 25.7785, -100.107]
["TAM", 22.2944, 97.8650]
["DM", 20.3218, -103.331]
["CJS", 31.4341, -106.429]
["COW", 22.8848, -96.8773]
["TIL", 32.3411, -114.97]
["HMO", 29.8959, -111.843]
["DME", 18.4537, -97.799]
["MID", 28.937, -89.6577]
["CTM", 18.5047, -88.2068]
["VER", 19.1459, -96.1873]
["GAY", 14.9999, -96.7254]
["MEX", 19.7251, -99.2554]
["ZIM", 17.6216, -101.463]
["PWR", 20.6801, -105.254]
["LIN", -12.8219, -77.1143]
["MAY", 22.9892, -82.4892]
["HMO", 4.70319, 74.1469]
["MIA", 25.7932, -80.2968]
["LAX", 33.9425, -118.408]
["JFK", 40.6398, -73.7789]
["TFC", 25.5483, -103.411]
["PWA", 15.8749, -87.8891]
["ACA", 16.7371, -99.754]
["MTT", 23.124, -106.068]
["GDL", 14.5435, -90.5279]
["DFW", 32.8948, -97.833]
["PHX", 33.4543, -112.812]
["PHL", 39.8719, -75.2411]
["CLT", 35.214, -80.9431]

victort: 30 Top
Beginning of buffer.

```

Figure 3: Ingresando un archivo csv con formato correcto

```

Actividades  Emacs  10 de sep 14:36
~term - GNU Emacs at victortorres

Captura de pantalla realizada
Puede pegar la imagen desde el portapapeles.

victort: ~/.../Proyecto1/WeatherProject git:( victor 0/2 7/1 x )
+ ls data/
dataset1.csv  datos.csv  historial.txt  nuevo.txt
victort: ~/.../Proyecto1/WeatherProject git:( victor 0/2 7/1 x )
+ python src/prueba.py
ME
["MTV", 25.7785, -100.107]
["TAM", 22.2944, 97.8650]
["DM", 20.3218, -103.331]
["CJS", 31.4341, -106.429]
["COW", 22.8848, -96.8773]
["TIL", 32.3411, -114.97]
["HMO", 29.8959, -111.843]
["DME", 18.4537, -97.799]
["MID", 28.937, -89.6577]
["CTM", 18.5047, -88.2068]
["VER", 19.1459, -96.1873]
["GAY", 14.9999, -96.7254]
["MEX", 19.7251, -99.2554]
["ZIM", 17.6216, -101.463]
["PWR", 20.6801, -105.254]
["LIN", -12.8219, -77.1143]
["MAY", 22.9892, -82.4892]
["HMO", 4.70319, 74.1469]
["MIA", 25.7932, -80.2968]
["LAX", 33.9425, -118.408]
["JFK", 40.6398, -73.7789]
["TFC", 25.5483, -103.411]
["PWA", 15.8749, -87.8891]
["ACA", 16.7371, -99.754]
["MTT", 23.124, -106.068]
["GDL", 14.5435, -90.5279]
["DFW", 32.8948, -97.833]
["PHX", 33.4543, -112.812]
["PHL", 39.8719, -75.2411]
["CLT", 35.214, -80.9431]

victort: 30 Top
Beginning of buffer.

```

Figure 4: Ingresando un archivo csv con formato correcto

6.2 Prueba una ciudad inexistente

¿Que pasa si ingresamos un nombre codigo de ciudad tanto de origen como de destino que no existe?

Si ingresamos un codigo (nombre acertado de la ciudad) de ciudad como ciudad de origen que no existe, el programa termina diciendo que la ciudad no existe

```
victort: ~/.../Proyecto1/WeatherProject git:( victor U:7 x )  
→ ^ python src/prueba.py  
mexico  
Ciudad de origen no disponible  
victort: ~/.../Proyecto1/WeatherProject git:( victor U:7 x )
```

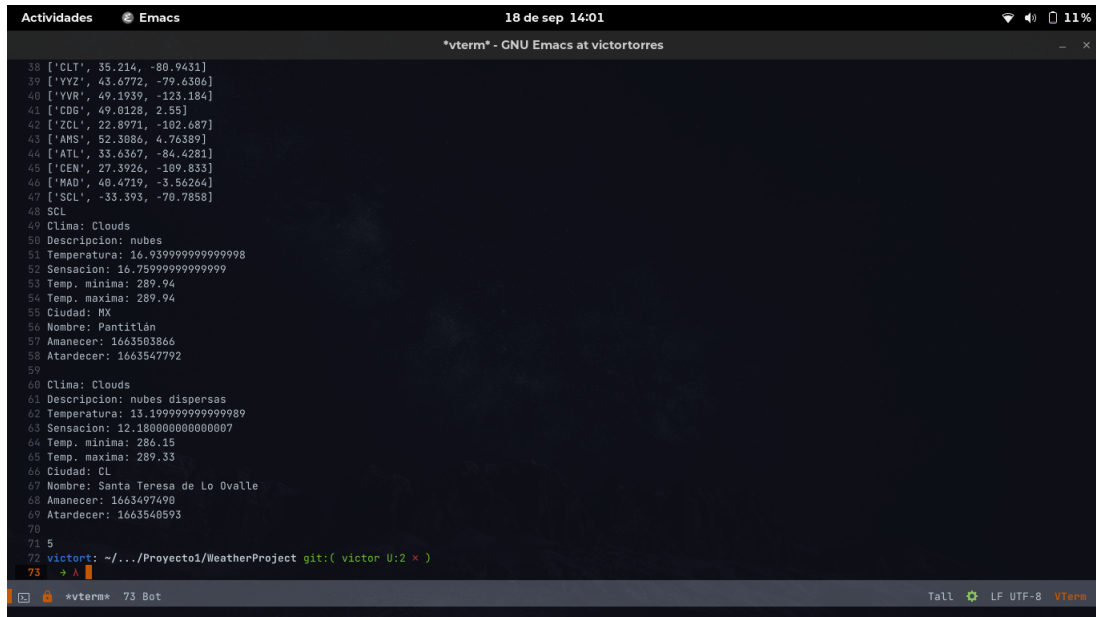
Figure 5: Ingresando una ciudad de origen inexistente

Si ingresamos un codigo de ciudad como ciudad de destino que no esta listada como destino disponible, el programa termina diciendo que esa ciudad de destino no existe.

```
[ 'MAD', 40.4117, -3.70264]  
[ 'SCL', -33.393, -70.7858]  
sls  
Ciudad de destino no disponible  
victort: ~/.../Proyecto1/WeatherProject git:( victor U:7 x )  
→ ^
```

Figure 6: Ingresando una ciudad de destino inexistente

Para ambos casos una prueba con entrada exitosa se ve asi:



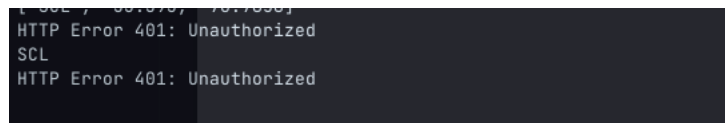
```
Actividades Emacs 18 de sep 14:01
*vtterm* - GNU Emacs at victortorres

38 ['CLT', 35.214, -80.9431]
39 ['YYZ', 43.6772, -79.6386]
40 ['YVR', 49.1939, -123.184]
41 ['CDG', 49.0128, 2.55]
42 ['ZLL', 22.8971, -102.687]
43 ['AMS', 52.3886, 4.76389]
44 ['ATL', 33.6367, -84.4281]
45 ['GEM', 27.3926, -109.833]
46 ['MAD', 40.4719, -3.56264]
47 ['SCL', -33.393, -70.7858]
48 SCL
49 Clima: Clouds
50 Descripcion: nubes
51 Temperatura: 16.939999999999998
52 Sensacion: 16.759999999999999
53 Temp. minima: 289.94
54 Temp. maxima: 289.94
55 Ciudad: MX
56 Nombre: Pantitlán
57 Amanecer: 1663503866
58 Atardecer: 1663547792
59
60 Clima: Clouds
61 Descripcion: nubes dispersas
62 Temperatura: 13.199999999999989
63 Sensacion: 12.180000000000007
64 Temp. minima: 286.15
65 Temp. maxima: 289.33
66 Ciudad: CL
67 Nombre: Santa Teresa de Lo Ovale
68 Amanecer: 1663497498
69 Atardecer: 1663548593
70
71 5
72 victor: ~/.../Proyecto1/WeatherProject git:( victor U:2 x )
73 + A
```

Figure 7: Prueba exitosa

6.3 Una API erronea

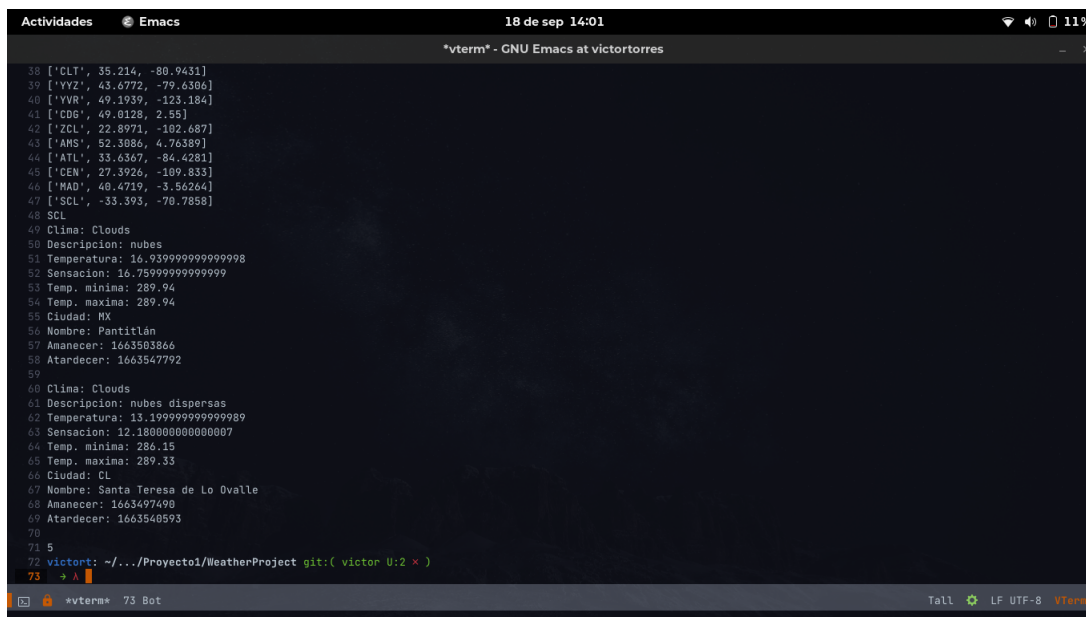
¿Que pasa si ingresamos una llave API no valida para la peticion de los datos? Si corremos el programa con una llave API erronea, el programa lanzara un mensaje el cual dice que la peticion esta no autorizada



```
['SCL', -33.393, -70.7858]
HTTP Error 401: Unauthorized
SCL
HTTP Error 401: Unauthorized
```

Figure 8: Haciendo una peticion con una llave API no valida

Si ingresamos una llave API valida:



```
38 ['CLT', 35.214, -80.9431]
39 ['YYZ', 43.6772, -79.6386]
40 ['YVR', 49.1939, -123.184]
41 ['CDG', 49.0128, 2.55]
42 ['ZLJ', 22.8971, -102.687]
43 ['AMS', 52.3886, 4.76389]
44 ['ATL', 33.6367, -84.4281]
45 ['GEM', 27.3926, -109.833]
46 ['MAD', 40.4719, -3.56264]
47 ['SCL', -33.393, -70.7858]
48 SCL
49 Clima: Clouds
50 Descripcion: nubes
51 Temperatura: 16.939999999999998
52 Sensacion: 16.759999999999999
53 Temp. minima: 289.94
54 Temp. maxima: 289.94
55 Ciudad: MX
56 Nombre: Pantitlán
57 Amanecer: 1663503866
58 Atardecer: 1663547792
59
60 Clima: Clouds
61 Descripcion: nubes dispersas
62 Temperatura: 13.199999999999999
63 Sensacion: 12.180000000000007
64 Temp. minima: 286.15
65 Temp. maxima: 289.33
66 Ciudad: CL
67 Nombre: Santa Teresa de Lo Ovale
68 Amanecer: 1663497498
69 Atardecer: 1663548593
70
71 5
72 victor: ~/.../Proyecto1/WeatherProject git:( victor U:2 x )
73 + A
```

Figure 9: Llave API valida

7 Costos

El costo de desarrollo por la beta seria de \$1000, el de desarrollo final seria un costo total ya de \$2000 y por darle mantenimiento habria un costo de \$500