Facultad de Ciencias, Universidad Nacional Autónoma de México

Modelado y Programación: semestre 3
Diego Arturo Velázquez Trejo y Luis Mario Escobar Rosales

API para Climas de tickets en vuelos comerciales

Definición del problema:
Día a día salen miles de vuelos de aeropuertos hacia otros estados o países. Por lo que se entrega una base de datos con información de boletos de vuelo con lugar de destino. Se necesita diseñar un programa que procese la información de los tickets y que genere un informe climático el mismo día que se corre el algoritmo.

Análisis del Problema:

Consideraciones:
Se necesita una aplicación que genere informe del clima de los tickets. Sin embargo, los boletos de avión se encuentran divididos entre dos bases de datos distintas, es decir, cada base de datos tienen información distinta. Es de importancia notar que coinciden en un dato conciso y necesario: **destino**.

-Análisis de los recursos : Dos bases de datos y API

Las bases de datos pueden o no contener datos fidedignos y tener en ellas registros inválidos para las APIs que regresan el estado climático. Además, la cantidad de datos puede acrecentarse y la API climática acepta como límite **60 peticiones por minuto**, por lo que se necesitará diseñar un sistema que sea capaz de contabilizar la cantidad de peticiones hechas de tal forma que no se genere un error.

Primeramente, notamos que las bases de datos contenían nombres de países inválidos, es decir, “México 68”, “28 de noviembre”, por ejemplo. Por otra parte, el usuario puede ingresar las coordenadas geográficas que hagan referencia al lugar de llegada. Sin embargo, las coordenadas pueden traer errores numéricos generando así, problemas al momento de solicitar su clima asociado.

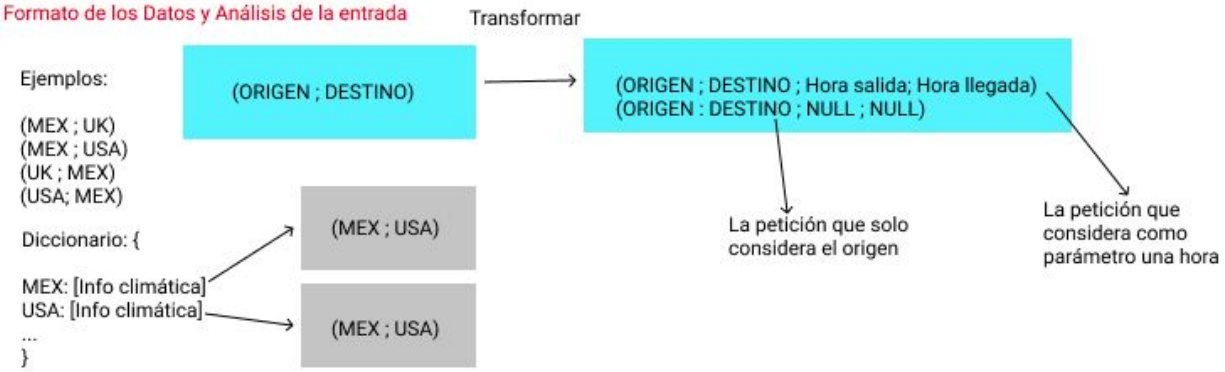
Secundariamente, los boletos de avión tienen como destinatarios un catálogo extenso de ciudades, en donde varias de ellas pueden llegar a repetirse. Acto seguido, el sistema tiene que identificar las repeticiones de dichas ciudades para evitar solicitar a la API el estado climático para el mismo lugar por 2 o más veces. Lo anterior será la justificación para el diseño de una metodología de filtrado y clasificación de información : caché. Es importante mencionar que no diseñamos un objeto caché como tal, sino que nos aprovechamos de la lectura de datos efectiva para la clasificación directa de los mismos.

Con esto nos percatamos que se pueden generar varios errores de entrada en los datos. A continuación mostraremos las preguntas de importancia:

1) ¿Qué importancia tienen las fechas de entrada y salida?
   No tienen importancia.
2) ¿Qué sucede con las fechas de salida? (¿Cuánto es la duración del clima, es decir, podemos quedarnos con el mismo clima de MEX para las 12:00 que para MEX a las 16:00 del mismo día?)
   El clima se considerará al momento que se consulta. Lo demás son datos ruido.
3) De acuerdo al siguiente formato de dato:
   (Origen : Destino : Longitud origen: Latitud origen : Longitud destino : Latitud destino)
   a) ¿Qué sucede si el origen está mal escrito pero la longitud y latitud no, y viceversa?
      Se hará la petición con el dato que esté en correcto formato. Si no funciona con la clave, se procede a hacer la petición con las coordenadas.
   b) ¿Qué sucede si las coordenadas (longitud y latitud) no existen?
      Si se puede obtener el clima con la clave IATA, se ignoran las coordenadas.
   c) ¿Qué sucede si no corresponde el destino con las coordenadas geográficas?
      Si se puede obtener el clima con la clave IATA, se ignoran las coordenadas.
4) ¿Qué sucede si las fechas de destino corresponden a un tiempo pretérito?
   Se considerarán las peticiones al momento en el que se realizan, ignorando las fechas y horas
5) (MEX : USA) & (USA : MEX) son vuelos distintos porque en uno MEX es origen y en otro es destino. Esto implica que el clima puede cambiar para ambos países al momento de partida que al momento de llegada:
   a) ¿Qué se tiene que considerar en ese caso?
      Al momento de realizar la petición se obtendrá sólo un clima por ciudad, haciendo la suposición de que este durará todo el día.
   b) ¿Se puede solicitar el clima para un momento dado y suponer que se mantendrá constante durante ambos vuelos?
6) ¿Qué sucede con los boletos que indican que el vuelo se realizará en fechas posteriores?
   Se considerará el clima para el momento en el que se haga la petición a la API.
7) ¿Puede pasar que las coordenadas geográficas cambien para MEX en distintos registros de boletos?
   Sí, sin embargo generalizaremos únicamente para un lugar.
8) ¿Qué dato se tomará como fidedigno (clave IATA o coordenadas geográficas)?
   La clave del país primero y si no funciona, se considerarán las coordenadas geográficas
9) ¿Las peticiones a la API de clima corresponderá para el momento en que se hagan o tendremos que considerar peticiones del clima para fechas pretéritas?
   Se considerarán únicamente para el momento en el que se realicen.

A continuación presentamos los esquemas que se realizaron para el análisis del problema:

Formato de los Datos y Análisis de la entrada



Incógnitas

Qué tanta importancia tienen las fechas de entrada y salida?
No importan. Asumimos que el pronóstico del clima dura un día entero.

Qué onda con las fechas de salida? (Cuanto es la duración del clima) Es decir, me quedo con el mismo clima de MEX para las 12:00 que para MEX a las 16:00?
No importa. Asumimos que el clima será el mismo.

Otra pregunta: Qué pasa si tenemos un dato de este tipo:
(ORIGEN ; DESTINO ; LONGITUD O ; LATITUD O ; LONGITUD D; LATITUD D)
Y el origen está mal escrito (lo que generará un error) pero la longitud y latitud están bien?

Qué pasa si las coordenadas (longitud o latitud) no existen?
Si primero probamos con la clave del país y funciona, entonces ignoramos la longitud y latitud.

Qué onda con las fechas de años pasados? Se ignora la fecha

(MEX ; USA) (USA ; MEX) Son distintas porque en una sales de México y llegas a USA mientras que en otra sales de USA y llegas a México. Ambos viajes implican que el clima puede cambiar para méxico o para usa (cuando sales y cuando llegas) , ahí qué tenemos que considerar? Podemos simplemente solicitar el clima para ese momento y entregarlo para ambos tickets prescindiendo de la hora o no?
Aquí vamos a hacer la petición para MEX, para USA y no nos importará lo demás. Solo serán 2 peticiones.

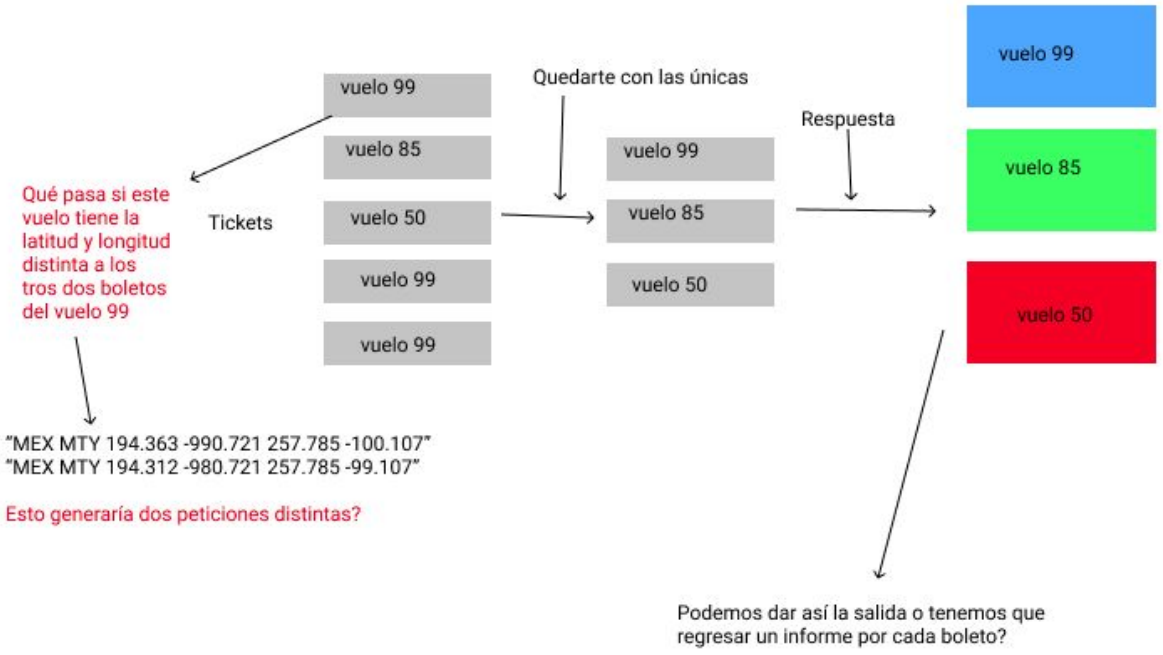
Para qué sirven las fechas?
Ruido

Observación: Una petición del clima para MX ahorita es una petición distinta a la petición del clima de MX en una hora o en dos
Esa observación no se cumple. Asumiremos que el clima es el mismo; la hora no importa.

Preguntas con respecto a las coordenadas. Puede pasar que para MX cambie la longitud o la latitud en distintos boletos?
Generalmente no cambiará (es caso particular de que primero chequemos la clave del país y si no, la long y lat).

Qué dato tomamos como el fidedigno?
La clave del país PRIMERO y si no funciona, tomamos la Longitud y latitud.

Una cosa es obtener las claves únicas de países y hacer las peticiones con esas claves y regresar la información (generalizada) para vuelo y otra cosa es iterar sobre toda la base de datos para ver los datos proporcionados POR BOLETO estén bien? Cuál es el camino que ellos quieren?
Sí vamos a tener que iterar, pero ya con un diccionario con los climas, para no tener que hacer todas las peticiones (repetidas).



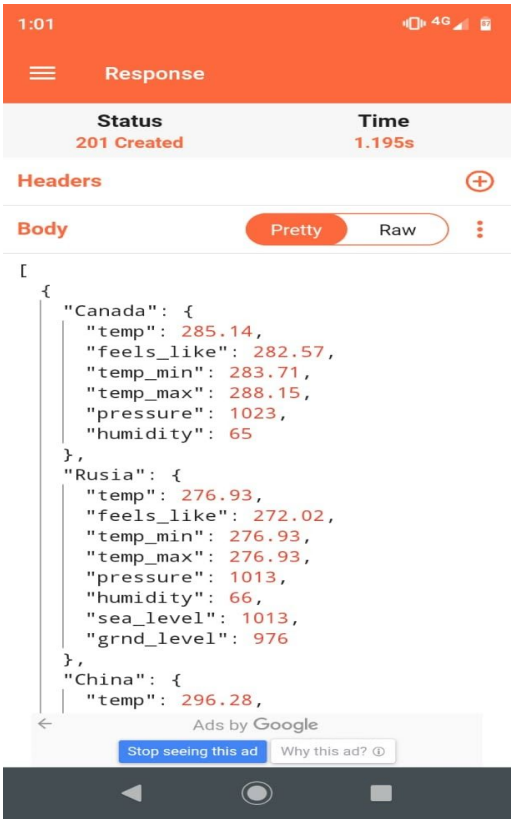
Selección de la mejor alternativa. ¿Qué tecnología se utilizó? Descripción de la Infraestructura del proyecto.

Se utilizará NodeJs, OpenWeatherMap, Javascript y tecnología Heroku con la finalidad de levantar un servicio web para su futura escalabilidad, además, pretendemos aprovechar los recursos de servicio web y microservicios con la finalidad de reducir el tiempo de respuesta de 16 minutos (como mínimo esperado) a 3 minutos.

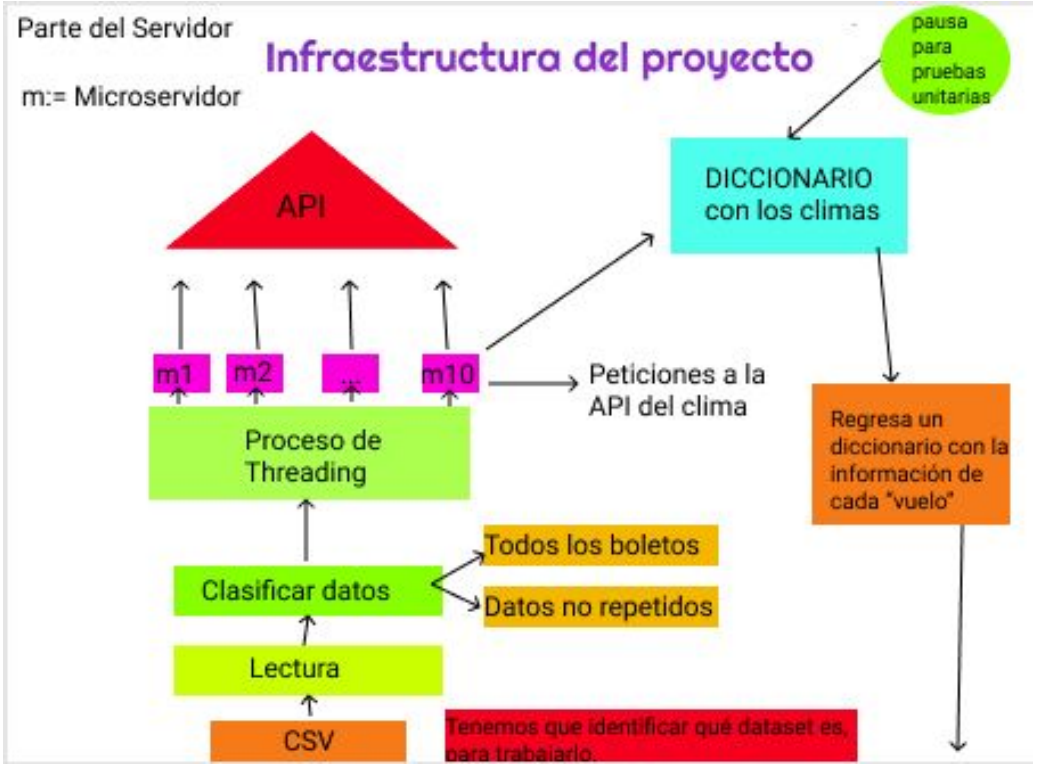
OPenWeatherMap es la API del clima que elegimos debido a su gran catálogo y flexibilidad que tiene respecto a las peticiones. Podemos solicitar peticiones por medio de los nombres de los lugares, varios tipos de claves (IATA, ISO), coordenadas, etcétera. Además, es de las pocas APIs de clima que aceptan peticiones de ciudades de países Latinoamericanos.

Vamos a aprovechar las ventajas de asincronía en procesos que nos brinda NodeJs. Sin embargo, es importante mencionar que NodeJs tiene poco con la implementación de la biblioteca “Workers” que le permite crear hilos multiprocesos en su código. En consecuencia, estos métodos son, por el momento, experimentales. Pero, posteriormente a la conclusión del proyecto, afirmamos que la decisión aumentó la eficiencia del programa o no se presentaron problemas algunos en la implementación dada.

Escalabilidad: la ventaja de nuestro diseño es que tenemos una API que es capaz de trabajar con grandes cantidades de datos en un intervalo de tiempo reducido. Acto seguido, se puede diseñar a futuro cualquier tipo de aplicación web o móvil que se conecten a nuestra API de servicios y que aproveche su rendimiento. A continuación, mostramos la funcionalidad de nuestro sistema desde una API Client APP de un celular android:



Finalmente, la tecnología web nos permite levantar microservicios y crear canales de conexión entre servidores en la nube para analizar con mayor rapidez particiones de los datos datos. A continuación mostraremos un esquema del sistema y pasaremos a describirlo:



Descripción general del sistema:

Nuestro sistema se levanta sobre tecnología NodeJs (Express), por lo que se instancia un servidor utilizando el comando "**npm start**" (Es importante mencionar que para la correcta ejecución del sistema se necesita tener instalado nodejs, node).

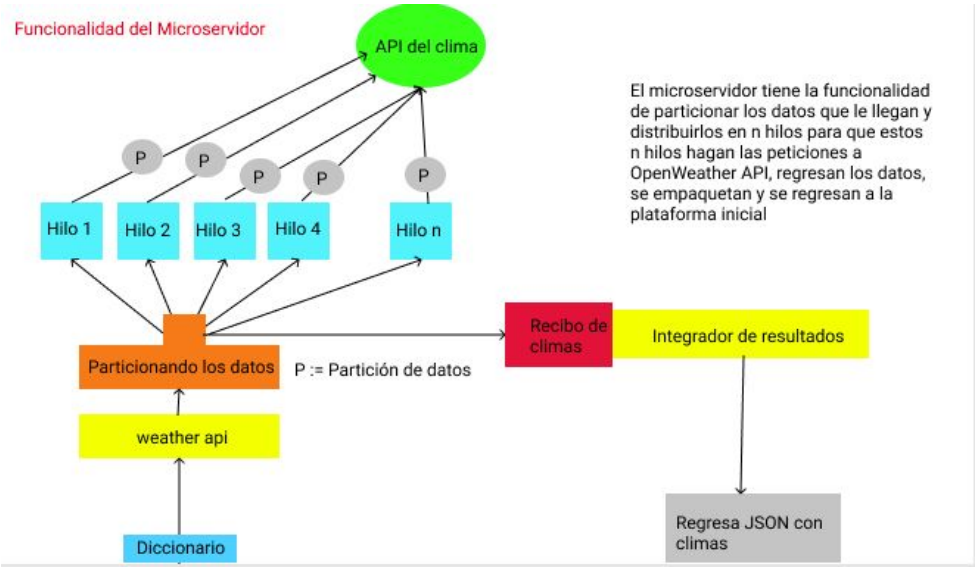
Primeramente, el sistema pasa a leer la base de datos que el usuario le proporcione. El proceso se realiza paralelamente, es decir, mientras lee los datos proporcionados en un archivo "**.csv**", éste identifica aquellos datos que presentan errores (prescindiendo de ellos) y también registra aquellos datos únicos, para la optimización de peticiones por ejecución efectiva.

Secundariamente, el sistema pasa a particionar el conjunto de datos (únicos y válidos: caché) en dos subparticiones. Como observación, el sistema puede únicamente analizar 1,100 claves **únicas** de países. Esto se debe a que contamos con 10 microservidores que realizan el análisis simultáneo de los datos y regresan la respuesta en un tiempo específico y dado. Acto seguido de particionar en dos conjuntos, cada conjunto será pasado a una función que generará una cantidad proporcional de hilos (como máximo 10) al tamaño de la partición, con la finalidad de que cada hilo realice una conexión a un microservidor distinto y le envíe un conjunto de datos de longitud (#partición)/55 con la finalidad de que este microservidor ejecute las peticiones a OpenWeather; independientemente al sistema.

Observación: Cada microservidor recibirá un paquete de datos no mayor a 55 claves/nombres de países.

Se reciben las respuestas de cada hilo (contienen las respuestas por servidor de las peticiones climáticas a OpenWeatherMap), y se regresan. Posteriormente, se leen los tickets y se le concatena a cada uno el clima de acuerdo a su clave IATA y si no existe en nuestra mini base de datos, de acuerdo a su coordenada.

Estructura del Microservidor:



Descripción de la Infraestructura del Microservidor:

El microservidor va a recibir como máximo un bloque de 55 peticiones, mismo que vuelve a re-particionar para volver a utilizar hilos; esta vez, cada hilo con el objetivo de realizar peticiones a OpenWeatherMap paralelamente.

Los hilos que se utilizan son tecnología Workers, proporcionados por la biblioteca estándar de NodeJs. En su utilización fue necesario usar la siguiente bandera: "**--experimental-worker"** dado que, como antes mencionamos, los hilos en NodeJs están en fase de prueba.

Direcciones URL de nuestros microservidores:

1) https://microservidora.herokuapp.com
2) https://microservidorb1.herokuapp.com
3) https://microservidor1233245.herokuapp.com
4) https://microservidorc.herokuapp.com
5) https://microservidorauxiliar123445.herokuapp.com
6) https://microservidorauxli123.herokuapp.com
7) https://microservidorcd123.herokuapp.com
8) https://microservidorlm21.herokuapp.com
9) https://microservidor1234peti.herokuapp.com
10) https://microservidorseccionb12.herokuapp.com

A futuro, ¿qué mantenimiento crees que podría requerir en un futuro?

Verificar que la plataforma sobre la cual los microservidores están montados (Heroku) siga con un buen y óptimo funcionamiento. Por otra parte, que la API sobre la que estamos haciendo las peticiones (OpenWeatherMap) siga realizando el mismo tipo de peticiones de clima utilizando el formato que nosotros definimos en el sistema.

Verificar que el funcionamiento de las bibliotecas utilizadas en NodeJs y Javascript sigan funcionando de igual forma, o de una manera similar, para evitar hacer cambios muy extensos en el programa

¿Cuánto crees que cobrarías por este pequeño reto y por futuros mantenimientos?

Consideraciones :

- Conocimiento de tecnologías de desarrollo web node js y javascript
- Diseño óptimo: Se esperaba que el programa tardará 16 minutos mínimo, en promedio 20 minutos (sin concurrencia); nuestro sistema es 10 veces más rápido.
- Conocimiento de tecnologías de la nube: implementación de microservidores en Heroku
- Escalabilidad: El producto puede seguir mejorando e implementando interfaces gráficas amigables, mayor robustez, etc.
- Código comentado y entendible: El código está comentado, es reutilizable y entendible.

- Tiempo y riesgo : Nos desvelamos haciendo cuentas de correo y microservidores, además de que casi somos baneados de OpenWeatherMap.
- Tiempo total de trabajo: 35 horas. Por lo que cobraremos $500 la c/h. Resultando un total de $17, 500 (**sin impuestos**).

Perfil de github en donde se puede encontrar el proyecto:

Infraestructura del sistema:
https://github.com/DiegoArturoVelazquezTrejo/Weather_api_Modelado/tree/diezMicroservidores
Infraestructura de los microservidores:

https://github.com/DiegoArturoVelazquezTrejo/MicroServidor_weather_api

Archivos Adjuntos (proceso de construcción y análisis del problema):