El problema del aeropuerto

Francisco Daniel Cruz Torres
316194099

Agosto 2019

0.1 Definición del problema

Un cliente quiere tener una aplicación' que le proporcione la información del clima de dos ciudades, una de origen y la otra de destino. Para esto la información entregada sera una cadena con el formato:

ORIGEN, DESTINO, LAT-ORG, LON-ORG, LAT-DES, LON-DES

El programa no necesita ser interactivo, y procesa mínimo 3000 peticiones.

0.2 Análisis del problema

El cliente necesita un programa que procese múltiples peticiones, y nos ha dado un archivo, donde cada linea representa una petición a ser respondida.
Necesidades:

- Manejar múltiples peticiones a la vez, dando favor al uso de concurrencia.
- Una API que brinde los recursos del clima en un tiempo definido.

0.3 Selección de alternativas

- Lenguaje y entorno de ejecución:
  Como lenguaje se ha optado por JavaScript, ya que este cuenta con concurrencia de tipo asíncrona de manera nativa y es orientado a eventos. Además usaremos Node.js como entorno de ejecución de JavaScript, ya que este no nos limita al entorno del navegador. Y al ser desarrollado en Node, esto permite la sustentabilidad a futuro al este poder ser integrado con un framework como express para tener una aplicación bajo el modelo cliente-servidor. Pero por ahora nuestra aplicación sera local, y cada linea representara una petición, de cualquier modo, sera concurrente, apelando a la rapidez y a la escalabilidad. Además, apelando a la interoperatividad, usaremos solo módulos estándar de Node.js, para no cargar el programa con dependencias innecesarias.

- Interfaz:
  Además al no ser interactivo por ahora, solo contará con una interfaz de linea de comandos (CLI), donde se mostraran las respuestas a cada una de las peticiones recibidas en el orden en el que sean procesadas.

- API del clima:
  Se ha decidido usar como API para obtener la información sobre el clima a la API de openweathermap, por su simpleza. Como limitación, es que esta solo nos permite realizar 60 peticiones por minuto lo que nos obliga a limitar el numero de peticiones concurrentes en nuestro programa.

0.4 ¿Como funciona el programa?

La prioridad es que todo sea concurrente (JavaScript solo soporta asincronía, no paralelismo), así si esto es montado en un servidor no habría ninguna operación que fuese capaz de bloquearlo.

Inicia el programa

- (0) Al inicio contamos con un cache de ciudades, un numero de llaves para API disponibles para hacer la petición (60 por default) y un numero de peticiones totales.
- (1) Lee el archivo linea por linea de manera asíncrona, esto es, que cada vez que se lea una linea, se active un evento para ir a la siguiente acción, que se espera también sea de naturaleza asíncrona; y cuando esta termine o se ponga en espera, continuamos leyendo lineas del archivo hasta que este se termine. Así no necesitamos leer todo el archivo antes de poder hacer las peticiones a la API, sino que las hacemos a la par.
- (2) Formatea la cadena de cada linea dentro de un arreglo asociativo de dos objetos de tipo ciudad con un id, latitud y longitud. Donde un elemento es el origen y otro el destino. Además las llaves del arreglo son los id's de tres letras asociadas a cada ciudad.
- (3) Recibimos el arreglo con ambas ciudades, y para cada ciudad en el array revisamos si esta no esta definida en el cache, si no lo esta, entonces hacemos la petición a la API para que se definan los valores de la ciudad. La API no responde hasta que haya llaves de la API disponibles, así nos aseguramos de que no sobrepasemos el numero máximo de peticiones permitidas. La petición a la API es de naturaleza asíncrona, pues no queremos que el servidor pare solo para esperar la petición. Cuando llegue la respuesta solo se dispara un evento, con el que definimos el valor de la ciudad en el cache y agregamos la ciudad a un array de retorno de las ciudades. De otra manera, si ya estaba en el cache, solo agregamos la ciudad al array de retorno. El array no regresa, hasta que los valores de ambas ciudades estén definidos, siendo esto también es asíncrono.
- (4) Una vez que se reciba el array con las ciudades respuesta, las imprimimos en un formato legible y las seguimos imprimiendo cada cierto tiempo.
- (5A) Asíncronamente refrescamos cada minuto el numero de llaves disponibles al default.
- (6A) Asíncronamente refrescamos cada 15 minutos los datos de cada ciudad dentro del cache.

0.5 Pensando a futuro

1. **Mantenimiento**
   A futuro, se podría mejorar el programa al este ser integrado como un servicio web, donde se hacen peticiones reales a un servidor. Ya que al manejar concurrencia el programa es capaz de soportar múltiples peticiones. El mantenimiento en si, seria la integración con mas servicios, para que este programa funcionase como un modulo libre disponible al publico.

2. **Costo del proyecto**
   Ya que el tiempo de desarrollo fue relativamente poco, pero el tiempo de documentación fue bastante, el costo seria:
   Unos $229.97 mxn y unos tacos de pastor (orden de 6).

3. **Costos del mantenimiento**
   Dependiendo de las necesidades del mantenimiento, el costo base serian unos $107 mxn y unos sabritones, mas extra dependiendo del tipo de mantenimiento.

4. **Pensamientos generales**
   El proyecto estuvo bien, el reto de hacerlo concurrente fue lo que le dio la dificultad real, ya que no estaba acostumbrado a hacer esto desde hace tiempo, sin embargo el hecho de usar concurrencia apoya a que el proyecto pueda ser escalado a mas a futuro.
   Además decidí no usar módulos diferentes a los estándar de Node, en pro de la compatibilidad, lo que al final hizo todo un tanto mas difícil, pero mas divertido. Aunado a esto decidí no usar promesas ni funciones async-await, sino solo callbacks, a la antigua, lo que fue una decisión horrible, pero he aprendido que a día de hoy y con las necesidades actuales, el uso de solo callbacks solo hace que el código sea un tanto mas enrevesado y se tengan que idear artimañas innecesarias para hacer cosas que con las promesas ya están resueltas.
   El tiempo estimado que tome en realizar este proyecto fue:

   - 2 hrs de modelado del problema.
   - 8 hrs de investigación sobre el lenguaje.
   - 2 hrs para programar lo principal.
   - 2 hrs para testing.
   - 4 hrs de documentación.

   No se si debería de incluir esto dentro del reporte, pero lo incluiré de todos modos, si hay algún problema, lo corregiré para futuros proyectos.
   Gracias.