Proyecto 01 Web Service

Universidad Nacional Autónoma de México

Facultad de Ciencias

Modelado y Programación

Integrantes:
Andrea Aguirre González
Hannia Laura López Ceballos
Mónica Miranda Mijangos

12 de octubre de 2021

Día a día miles de personas van y vienen de aeropuertos a aeropuertos, cambiando luego drásticamente de zona horaria y clima. El aeropuerto de la Ciudad de México 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 día que se corre el algoritmo. No es interactivo, solo nos interesa el clima.

1. Desarrollo

1.1. Definición del Problema

- ¿Qué es lo que queremos obtener? El informe del clima de la ciudad de salida y la ciudad de llegada para 3 mil tickets que salen el mismo día.
- ¿Cuáles son los datos que tenemos para obtenerlo? Una archivo csv proporcionado por el aeropuerto con 3 mil tickets los cuales tienen 6 campos con los datos correspondientes al nombre y las coordenadas de las ciudades de origen y destino.
- ¿Son suficientes? Sí, para conocer el reporte del clima de una ciudad sólo debemos conocer su nombre o sus coordenadas.
- ¿Qué hace que el resultado obtenido resuelva el problema? ¿Cuál es la característica que hace de un resultado, una solución? El personal de aviación puede estar al tanto de los cambios drásticos del clima de las ciudades de las que partirá y de las ciudades que visitará en un mismo día, para que pueda tomar medidas en caso de cambios metereológicos muy extremos.
- ¿Qué operaciones o constucciones se deben obtener para llegar a la solución? Leer la base de datos y hacer una solicitud a los aeropuertos acerca del clima, al momento, en la ciudad para después entregar dicho informe para 3000 tickets.

- Entrada / Salida
  - Tipo: String
  - Formato: CSV / Texto en terminal
  - Tamaño: 3001 filas / reporte con 5 campos por ticket
  - Cantidad: 3000 tickets / 6000 informes
  - Fuente / Diseño: Recolección de datos del Aeropueto / Tabla de información ordenada por campos especificados por el cliente

1.2. Análisis del Problema

- Requsitos funcionales:
  - ¿Qué debe entregar como salida dado cierto tipo de entrada? Dado un ticket con datos de origen y de destino debe regresar el reporte del clima para la ciudad de origen y la ciudad de destino.
  - ¿Qué se debe llevar a cabo? Al registrar un ticket, se debe mandar una soliciud a la API de clima elegida, la cual debe regresar el reporte del clima de la ciudad de origen y de destino.
- Requsitos no funcionales:
  - Eficiencia

    El sistema debe ser capaz de procesar las solicitudes que se le hacen a OpenWeather sin saturarse, esto lo lograremos mediante un método el cual le dira cuando procesar únicamente una solicitud en caso de que estas sean repetidas, lo que gastara menos peticiones y por lo tanto evitara saturarse. Los datos modificados en la base de datos deben ser actualizados para que el sistema tenga y pueda imprimir la información correcta al usuario.
  - Tolerancia a fallas

    El programa termina de ejecutarse aún cuando falle algún elemento del sistema, en este caso, un ejemplo sería que no se proporcionara bien la información al procesar la solicitud, a lo que el programa le respondería con el mensaje "No puedo procesar los datos".
  - Amigabilidad

    En esta parte únicamente se solicita al usuario introducir los datos requeridos para poder procesar su solicitud. El informe que se imprime al realizar la petición es una descripción clara sobre el estado del tiempo de las ciudades de origen y destino, por lo tanto la podemos considerar como una interfaz amigable.
  - Escalabilidad

    Influye en el rendimiento del sistema de forma significativa, asimismo, supone un factor crítico en el crecimiento de un sistema dada su capacidad de adaptación y respuesta con respecto al rendimiento del mismo.
  - Seguridad

    Los permisos o en esto caso la llave para poder hacer una petición únicamente puede ser cambiada por los administradores.

- Interoperatibilidad

  Permite que los usuarios identifiquen los usos autorizados para las plataformas y los contenidos en una red interoperable, para poder lograr el intercambio, deben contar con características técnicas y de estructura específicas que faciliten la comunicación.

1.3. Selección de la mejor alternativa

- Paradigma de programación: Orientado a Objetos.
  Abstraeremos las características y comportamiento de cada ticket, desde cómo lo obtenemos y realizamos una llamada a nuestro web service hasta como devolvemos esos valores.

- Lenguaje: Python.
  Ya que es un lenguaje multiparadigma enfocado en la legibilidad del código, además de ser interpretado, es decir, puede compilar el código fuente original en una forma intermedia más compacta, y después traducir eso al código de máquina; dinámico, las variables pueden tomar valores de distinto tipo, y multiplataforma.

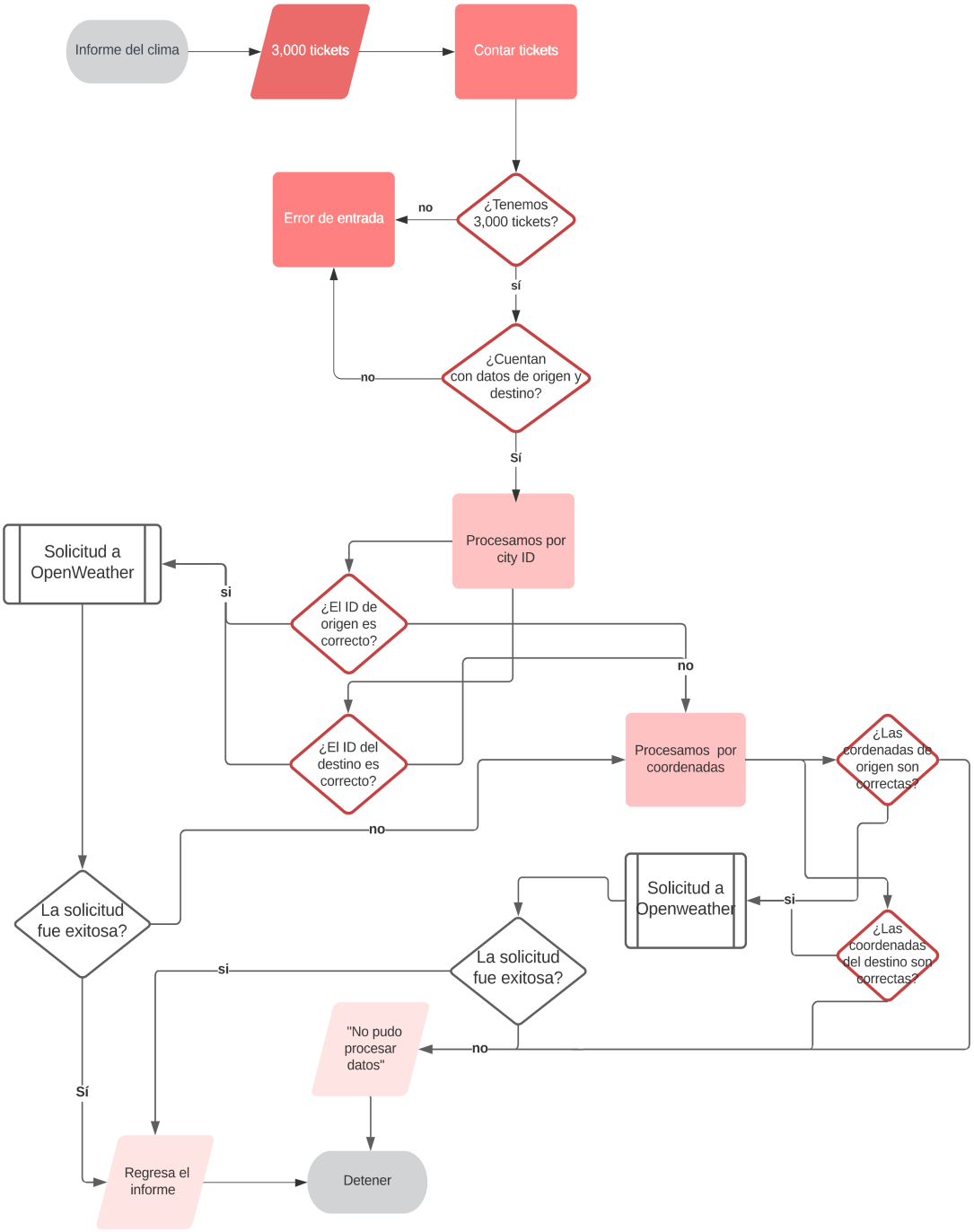
- Herramientas para pre o post precesar datos: web services, OpenWeather.

- Frameworks o bibliotecas:

  - csv - para llamar a DictReader y mapear la información de cada fila a un diccionario.
  - sys - generamos la excepción SystemExit con el que Python sale sin imprimir el seguimiento de pila.
  - tabulate - para imprimir tablas de manera estética.
  - request - para realizar las solicitudes HTTP.
  - json - para trabajar con la respuesta de la API.
  - time - nos permite introducir un retraso en la ejecución de nuestro programa.

  - ¿Qué tan útiles son? OpenWeather nos permite obtener la información del clima de cada ciudad fácilmente con una llave que podemos obtener de manera gratuita.
  - ¿Qué se necesita para usarlas? importar sus respectivos módulos, además de instalar la paquetería de pip tabulate para poder imprimir nuestros reportes.
  - ¿Están bien documentadas? Sí
  - ¿Han sido probadas? Sí
  - ¿Vale la pena usarlas? Definitivamente, nos ahorra tener que hacer nuestras propias biliotecas. O bien la necesidad de generar alternativas que hagan peticiones a http, o en el peor de los casos que soliciten y obtengan la información del clima cada aeropuerto o ciudad requerida.
  - ¿Son seguras? Sí

1.4. Diagrama de flujo



1.5. Mantenimiento

- ¿Cuánto cobrarías por el proyecto y futuro mantenimiento? Para poder poner un precio al proyecto podemos desglosar una lista de tareas a traves de las cuales se estimara el tiempo que se le dedicara a cada una en horas, hay que tener en cuenta que a cada una de las tareas se le deben agregar los costos que se requerirán para que el sistema funcione, como lo sería:

  - Equipo específico para el proyecto.
  - Servidores en la nube.
  - Dominios
  - Licencias
  - Bases de datos

- Una buena forma de determinar el precio por hora del proyecto es tomando en cuenta la experiencia que se tiene y en todo caso lo que se desea ganar mensualmente. Tomando esto en cuenta, de acuerdo a nuestra experiencia y horas trabajadas, el precio por hora sería de 168.75 MXN lo que nos da un total de 27,000 MXN.

- Tomando en cuenta de que el programa no es interactivo y se desea que se mantenga de esa manera, su mantenimiento es relativamente fácil. Las cosas a considerar serían el número de tickets que obtiene, ya sea un número mayor o menor al que fue establecido en este caso en particular, al igual que la expansión de la base de datos en caso de que se agregué un destino nuevo a la aerolínea.
  Si es que se quisiera implementar algún tipo de acción donde el usuario tenga que interactuar con el programa, ya sea la opción de escoger un ticket específico dentro de la base de datos entonces ahí se tomarían en consideración las distintas formas que puede lleverse esto acabo: creando una lista con los tickets dados, hacer que el usuario escriba el city ID o las coordenadas geográficas.
  También hay que considerar el funcionamiento de el web service. En esté caso en particular se utilizó la versión gratis de OpenWeather, este tiene restricciones y si es que se quisiera ampliar el packete de servicios de este se tendría que considerar pagar por el servicio deseado o buscar un servicio web alternativo.

1.6. Cambios Sugeridos

- Los cambios realizados fueron los siguientes:
  - El dataset se coloco fuera del main en la carpeta src/data.
  - Se elimino el código basura en tickets.py
  - Las pruebas de peticiones fueron documentadas.
  - Se incluyó 1 prueba más, la de datos, que igualmente ya está documentada.
  - Se específico el uso de tabulate y de pruebas uniatrias en el Readme.
- Con estos cambios se espera que mejore el entendimiento de como hacer correr el proyecto, así como mejorar su eficiencia.