UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Proyecto 1

Modelado y Programación

Autores:
Aguirre Muñoz Leonardo
318017686
wraithpuma@ciencias.unam.mx
Valencia Cruz Jonathan Josué

12 de octubre de 2021

1. Definición del problema

Dada una lista de 3000 vuelos con ciudad de origen y ciudad de destino, se solicita que se realice un programa que de un informe del clima en tiempo real de la ciudad de salida y la ciudad de llegada para cada uno de los vuelos. El programa no debe ser interactivo

Los datos del clima requeridos en el informe son los siguientes:

a) Descripción del clima

b) Temperaturas mínima y máxima

c) Humedad

d) Sensación térmica

e) Presión

2. Análisis del problema

Se nos da un archivo csv que almacena la siguiente información:

a) Ciudad de origen

b) Ciudad de destino

c) Longitud y latitud de origen

d) Longitud y latitud de destino

A partir de esta información se solicita que se entregue la información anteriormente dicha respecto al clima, esta información se va extraer de alguna API dedicada al clima y tiempo atmosférico, en este caso se hará uso de la API OpenWeatherMap, siendo que para su funcionamiento, requeriremos de algunos datos los cuales son:

a) Una Key personalizada por parte de OpenWeatherMap (Esta se solicita desde la página oficial de OpenWeatherMap, creando una cuenta de usuario)

b) Información de la ciudad o lugar a realizar la petición del informe del clima.

c) Lenguaje a solicitar la descripción del clima (En este caso español)

d) Unidades de medida para las diferentes variables del clima tales como temperatura y humedad (En este caso haremos uso del Sistema de Unidades Internacional)

Una vez identificado y obtenido la información requerida para hacer uso de la API, procedemos a realizar las llamadas a la API almacenando la información obtenida en algún lugar y comenzamos a vincular el informe del clima obtenido para su respectiva ciudad de origen y ciudad de destino. Esta información pasa a ser mostrada en pantalla o en algún otro medio en donde se pueda visualizar el contenido.

La cuestión es la siguiente, dado que nuestro número de solicitudes a la API está restringida a un cierto límite, debemos hacer uso de un método que nos ayude a evitar realizar solicitudes repetidas, es decir, podemos hacer uso de alguna memoria caché o por defecto filtrar la lista de ciudades de modo que no haya ciudades repetidas.

Otro problema a solucionar son los errores que pueden presentarse al momento de realizar una solicitud a la API, estos errores son diversos pero algunos de ellos son:

a) Error 404, este error surge cuando la información solicitada no se encuentra y esto puede deberse a diversos motivos como lo es tener problemas de conectividad.

b) Ciudad no encontrada, este es parte del error 404 pero quisimos mencionarlo debido a que es un problema a considerar en primera instancia.

c) Tiempo de solicitud excedida, este error puede surgir por ciertos motivos como lo es problemas de conectividad o se realizan varias peticiones de manera simultanea.

Para solucionarlo, se opta por un manejo de errores implementado en el programa llamados excepciones que, en caso de encontrar algún error, pasa a la siguiente solicitud.

3. Selección de la mejor alternativa

Se utilizó el lenguaje de programación en R ya que simplifica y aligera el trabajo de lectura, almacenamiento y modificación de extensas bases de datos además de implementar bibliotecas que ayudan en la lectura y escritura de documentos csv, siendo que el problema requiere de su uso.

Otra virtud que posee el lenguaje R es su fácil uso, lenguaje práctico y sencillo, haciendo que el proceso de aprendizaje de este lenguaje no sea un obstáculo al momento de querer realizar el proyecto.

De igual manera se usó la API OpenWeatherMap pues además de proporcionarnos la información requerida por el problema resulta fácil de usar y no requiere de un gran esfuerzo para comprender el funcionamiento de la misma, a su vez puedes modificar la información obtenida a partir de la misma como lo podría ser pasar de kelvin a grados centígrados o grados farenheit e inclusive cambiar el idioma.

También se hizo uso del formato JSON para la solicitud de información hacia la API pues resultaba fácil de cambiar el formato a csv por la biblioteca de r "jsonlite", siendo que, tanto el formato JSON como el formato csv, hacen del trabajo de lectura, escritura y manipulación de contenido algo más sencillo.

Debido a que se tenían inconvenientes con las llamadas a partir de las coordenadas de los aeropuertos, se optó por buscar una lista en internet la localización de los aeropuertos junto a su abreviación, puedes consultar esta información en el siguiente enlace: https://www.aeropuertos.net/america-norte/mexico/

4. Diagrama de flujo o pseudocódigo (pueden hacer cualquiera de las dos o las dos) y explícalo.

```
Clase VuelosFiltrados{

#Función que lee el archivo de los vuelos
leerarchivo(){

    #Devuelve la variable con la información leída del archivo
    archivo = leer.csv("Filtros/dataset1.csv")

    #Devuelve la variable archivo
    regresar(archivo)
}

#Filtra el archivo usando solo los vuelos en origen y destino
filtrarviajes(arc){

#Filtra la información obteniendo las ciudades de los vuelos en origen y destino
filtro1 <- leer(arc,["origin", "destination")])

#Devuelve la variable con la información mencionada
regresar(filtro1)
}

#Función que filtra los vuelos repetidos
filtrarVuelosRepetidos(archi){

#Revisa que cada renglón de las columnas con "origin" y "destination" si están repetidas
filtro = leer(arc,["origin", "destination")])

#En caso de ser así los elimina de manera que solo quede 1
#(Distinto filtra las coincidencias que no se repiten entre dos listas)
ciudadesSinRepeticion <- distinto(origin,destination)

#Devuelve la variable con los cambios realizados
regresar(ciudadesSinRepeticion)
}
```

```
#Función que filtra todos los vuelos tanto de
#origen como destino para facilitar la petición del clima
unirFiltrarLista(ciudadesSinRepeticion){

#Convierte la tabla en una lista de manera que se obtendrán dos listas
ciudadesSinRepeticion = lista(ciudadesSinRepeticion)

#Concatena los elementos a manera de hacer una única lista
listavuelos = concatena(ciudadesSinRepeticion$origin,ciudadesSinRepeticion$destination)

#Ahora pasa los datos de la lista a manera de una tabla
listabuena = Tabla("Ciudad"=listavuelos)

#Usando la tabla se realiza de nuevo un filtro
#que los elementos repetidos solo se muestren una vez
vuelossinrep = listabuena(quitarDuplicados(Ciudad))

#Lee un archivo donde viene la información de los aeropuertos
aeropuertos = leer.csv("Filtros/Aeropuertos.csv")

#Número de renglones que tiene el archivo de aeropuertos
a = longitud(aeropuertos)

#Número de renglones que tiene el archivo de ciudades
b = longitud(vuelossinrep)

#Agrega una columna a las ciudades sin repetidos con el nombre Detalles
vuelossinrep = agregarColumnas(vuelossinrep,"Detalles")

for(i desde 1 hasta b){
for(j desde 1 hasta a){
#Revisa si la abreviación está en el archivo de los aeropuertos
Si(aeropuertos$Abreviatura[j] == vuelossinrep$Ciudad[i]) entonces{

#Asigna el espacio en Detalles el nombre del aeropuerto
vuelossinrep$Detalles[i] = aeropuertos$Ubicación[j]
}
}
}

#Quita los espacios que tenga y los remplaza con %20 para realizar las peticiones
vuelossinrep$Detalles <- reemplazar( " ", "%20", vuelossinrep$Detalles)

#Guarda los datos en un archivo csv.
escribir.csv(vuelossinrep,"Filtros/VuelosDescripcionFiltrados.csv")

#Devuelve la variable con la nueva información
regresar(vuelossinrep)
}

#Funcion que crea una tabla para poder añadir la informacion del clima de cada ciudad
#Parametros
#Vuelos Es el archivo que tiene la informacion de la anterior función
crearTablaClima(Vuelos){

#Añade 6 columnas para poder almacenar los datos del clima una vez realizada la petición.
vuelosSinDatosClima = agregarColumnas(Vuelos,["Descripción","Temp.min","Temp.max",
"Sens.Térmica","Humedad","Presión","Ciudad"])

#Regresa la variable con las modificaciones para almacenar los datos al realizar la petición
regresar(vuelosSinDatosClima)
```

```
}

#Función que obtiene realiza la petición para
obtener los datos del clima

obtenerDatosClima(Aviso)
{
#Alerta de un aviso en caso de una advertencia o de un error
Si (Aviso == "error" ó Aviso == "error" ) entonces{
respuesta = "Hubo un error"
regresar(respuesta) }

#En caso de no haber error, se guarda la petición en un archivo JSON
En otro caso{GuardarJson(aviso)}
}

}

Clase LokiWeatherMap{ (Se trata de la clase principal)

#Variable que guardará el archivo leído
archivo = leerarchivo()

#Variable que guarda todos los vuelos de origen y destino
filtrados = filtrarviajes(archivo)

#Variable que guarda las modificaciones del archivo sin #repeticiones
VuelosSinRepetir = filtrarVuelosRepetidos(archivo)

#Variable que guarda las modificaciones para realizar las #peticiones
ListaFiltrada = unirFiltrarLista(VuelosSinRepetir)

#Variable que añade las columnas
TablaClima = crearTablaClima(ListaFiltrada)

#Variable que guarda el numero de renglones
tamano = longitud(TablaClima)

#Iterador para realizar cada petición de cada ciudad para #almacenarlo en la tabla
for(i desde 1 hasta tamano){
    imprimir(concatenar("Obteniendo datos del
    clima...",i,"de",tamano))

#Variable que guarda el enlace con el nombre de la ciudad para #realizar la petición
Clima = concatenar("http://api.openweathermap.org/data/2.5/weather
?q=", TablaClima$Detalles[i],"&appid=7e10bbf74759c0ce960752bd913f4
223&lang=es&units=metric")

#Función que realiza la petición y envía un mensaje si existe una
#advertencia o error, en caso contrario sigue
revisar = tryCatch({
DatosClima = obtenerDatosClima(Clima)
},
warning(advertencia){
imprimir(concatenar("Cuidado: ", advertencia))
},
error(error){
imprimir("Error: Obteniendo el clima del siguiente... ")
},
finally(seguir)
{})
```

```
#Guarda los datos obtenidos de la petición
Clima_filtro = unir(DatosClima[["main"]],DatosClima[["weather"]][["description"]])

#Se rescatan los datos solicitados para descartar cualquier
#información ajena a lo solicitado.
Clima_filtro$temp=NULL
Clima_filtro$sea_level=NULL
Clima_filtro$grnd_level=NULL

#Almacena los datos en la tabla de clima del respectivo lugar
#Descripción del clima si es nuboso, soleado, lluvioso...
TablaClima$Descripcion[i] = Clima_filtro$y[1]

#Temperatura mínima del lugar
TablaClima$Temp.min[i] = Clima_filtro$temp_min[1]

#Temperatura máxima del lugar
TablaClima$Temp.max[i] = Clima_filtro$temp_max[1]

#Sensación térmica del lugar
TablaClima$Sens.Termica[i] = Clima_filtro$feels_like[1]

#Indica la Humedad del lugar
TablaClima$Humedad[i] = Clima_filtro$humidity[1]

#Señala la presión del lugar
TablaClima$Presion[i] = Clima_filtro$pressure[1]
}

#Guarda la información capturada en un archivo csv.
escribir.csv(TablaClima,"Filtros/CiudadesDatosClima.csv")

#Agrega columnas para poder agregar los datos solicitados para
#conocer el estado del tiempo de la ciudad de salida
filtrados = agregarColumnas(filtrados,"Descripcion.Origen","Temp.min.Origen"
,"Temp.max.Origen","Sens.Termica.Origen","Humedad.Origen"
,"Presion.Origen","origin.Origen")

#Agrega columnas para poder agregar los datos solicitados para
#conocer el estado del tiempo de la ciudad a arribar
filtrados <- agregarColumnas(filtrados,"Descripcion.Des","Temp.min
.Des","Temp.max.Des","Sens.Termica.Des","Humedad.Des","Presion.Des","destination")

#Variable que guarda el numero de vuelos que se tienen
NumerodeVuelos = numeroColumnas(filtrados)

#Se realiza la asignación de los datos en su respectivo campo
#antes de poder mostrar en pantalla
for(i desde 1 hasta tamano){
for(j desde 1 hasta NumerodeVuelos){

Si(TablaClima$Ciudad[i]==filtrados$origin[j]) entonces{
filtrados$Descripcion[j] = TablaClima$Descripcion[i]
filtrados$Temp.min[j] = TablaClima$Temp.min[i]
filtrados$Temp.max[j] = TablaClima$Temp.max[i]
filtrados$Sens.Termica[j] = TablaClima$Sens.Termica[i]
filtrados$Humedad[j] = TablaClima$Humedad[i]
filtrados$Presion[j] = TablaClima$Presion[i]
}
En caso de (TablaClima$Ciudad[i]==filtrados$destination[j]) entonces{
```

```
filtrados$Descripcion.Des[j] = TablaClima$Descripcion[i]
filtrados$Temp.min.Des[j] = TablaClima$Temp.min[i]
filtrados$Temp.max.Des[j] = TablaClima$Temp.max[i]
filtrados$Sens.Termica.Des[j] = TablaClima$Sens.Termica[i]
filtrados$Humedad.Des[j] = TablaClima$Humedad[i]
filtrados$Presion.Des[j] = TablaClima$Presion[i]
}
}

}

#Se guarda el archivo para poder tener un respaldo

escribir.csv(filtrados, "Filtros/VuelosconClima.csv")

#Se muestra la información solicitada de los respectivos vuelos
imprimir(filtrados)

}
```

5. Mantenimiento y proyecto a largo plazo

A lo largo de la creación de este proyecto se tuvieron algunas complicaciones y se tiene en mente algunas cuestiones que podrían hacerse a futuro como parte de dar mantenimiento al programa, algunos de estos planteamientos son:

a) Una de las cosas a mejorar a corto plazo es la parte de implementación de una interfaz gráfica o de algún medio de visualización más agradable para el usuario, esta interfaz ayudaría para hacer del programa más amigable con el usuario.

b) Otra cuestión a remarcar es la implementación de la opción de actualizar el programa para así actualizar el estado del clima en las diferentes ciudades de origen y destino de los vuelos, la razón es simple y es que en una situación en la vida real un programa como este requiere de tener información lo más actualizada posible debido a que es de utilidad para diversos aeropuertos del mundo y pues, si el programa lo implementara sería un gran acierto.

c) El tercer posible mantenimiento que podría requerir el programa sería la parte de verificar actualizaciones, ya sean por parte del lenguaje de programación, de la API o incluso del mismo sistema operativo pues en el caso de la API podría suponer un problema mayor para el programa pues si se modificase algún apartado para su funcionamiento, haría de nuestro programa obsoleto y es por eso que sería un importante punto a considerar.

d) El cuarto es la actualización del código para atender complicaciones que puedan surgir por parte de los usuarios pues suele suceder que cuando se crea un programa, no se tiene en consideración algunos aspectos que pueden hacer de errores para algunos usuarios, algunos de estos errores pueden ser bugs, situaciones no planeadas por parte del programa, errores de compatibilidad con algún equipo por parte de algún usuario, entre otros.

e) El quinto y último punto a mencionar pero no menos importante es el mejoramiento y optimización de código de manera general pues al ser un proyecto creado a partir de programadores con poca experiencia y en formación da lugar a que el programa sea poco eficiente en algunos aspectos y que se podrían mejorar con la implementación de algunas metodologías.

Una vez dicho lo anterior, sobre la cuestión de cuánto se esperaría cobrar por la realización de este programa y por futuros mantenimientos, una situación a plantearse consiste en la labor de un programador, que presentan varios retos, siendo principalmente el planteamiento y desarrollo en la solución de un problema un aspecto clave a considerar al momento de estimar algún honorario además de quien solicita el servicio del programador, horario de trabajo, fecha de entrega, factores externos como lo es consultar el servicio de demás lugares, el tipo de problema a resolver; esto formula la "pregunta definitiva"de responder pero diremos que, en este caso, se cobraría por el programa un monto de alrededor de 1000 a 4000 pesos aproximadamente pues se trata de un proyecto menor y que no requiere de una gran planificación de por medio.

6. Cambios realizados recientemente

Desde que se entregó el proyecto por última vez, se realizaron los siguientes cambios:

a) Se hicieron cambios en el nombre de las variables, de modo que sus nombres sean más claros y den una mejor descripción de su funcionamiento.

b) Se añadieron cambios a la salida del programa, se presentaban problemas con la impresión de la tabla con los datos del clima.

c) Se reestructuró la organización de los archivos contenidos en OpenWeatherMap/, la carpeta llamada 'test' se renombró a 'LokiWeatherApp', se creó la carpeta filtros y se movieron los archivos csv a esta carpeta.

d) Se renombraron los archivos 'test_mi_codigo.R' y 'micodigo.R' por 'LokiWeatherApp.R' y 'Vuelos-Filtrados.R'.

e) Se agregó el apartado 'Cambios realizados recientemente', además de agregar un párrafo más en la sección 'Selección de la mejor alternativa', explicando la razón de 'filtros'.

f) Se modificó en su totalidad el pseudocódigo agregado en la sección 'Diagrama de flujo o pseudocódigo (pueden hacer cualquiera de las dos o las dos) y explícalo'.

g) Se añadió contenido al archivo 'README.md', en él se muestra cómo ejecutar el programa además de información adicional de cada archivo ubicado en el proyecto.

Puedes ver los cambios más a detalles en el apartado Commits en el GIT del proyecto:

https://github.com/WraithLion/OpenWeatherMap/commits/main