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 aereopuerto 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.

4 Pseudocodigo

```
for cada vuelo do
    if ciudad de origen no esta registrada then
        registrala en la primera casilla vacia
    end if
    if ciudad de destino no esta registrada then
        registrala en el arreglo de la ciudad de origen
    end if
end for
for ciudad de Origen dociudades
    Escribir en cada json el arreglo de ciudad Origen
end for
Leer ciudad Origen
Mostrar todos los destinos posibles dada la ciudad de origen
climaOrigen = obtenerClima(ciudadOrigen, api)
Leer ciudad destino
climaDestino = obtenerClima(ciudadOrigen, api)
Mostrar Clima
Registrar el vuelo en un archivo
function obtenerClima(ciudadOrigen, api)
    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

6 Pruebas

6.1 Prueba con el archivo csv

6.2 Prueba una ciudad inexistente

6.3 Una API erronea