**Proyecto 01**
**Semestre 2021-1**
Prof. José Galaviz Casas
Ayud. María Ximena Lezama
**Modelado y programación**

Kevin Ariel Merino Peña[1] Armando Abraham Aquino Chapa[2]
9 de octubre de 2020

# 1. Definición del problema

# 2. Análisis del problema

**Análisis del problema y selección de la mejor alternativa.** Al ir analizando los diversos rubros que puede abarcar el problema, se determinó que lo más adecuado para resolverlo era obtener las siguientes clases:

- **Clase Weather:**.
  - Esta clase sería la encargada de realizar las distintas peticiones al servidor para obtener todos los datos correspondientes al clima.
  - Manejar los posibles errores, como el exceso de numero peticiones en un lapso de tiempo, para no obtener ningún problema con el servidor,y manejar los errores donde no es posible consultar la información
  - Otorgar de un formato correcto y legible a la salida del programa
- **Clase CSVReader**
  - Su función principal sería leer los archivos *csv* otorgados por el aeropuerto de la Ciudad de México.
  - Preprocesar cierto tipo de información que si es esencial de la que es redundante o no tiene utilidad para nuestros objetivos.
  -

# 3. Selección de la mejor alternativa

# 4. Pseudocódigo

## CSVReader.py

**Función 1:** read_no_repeated_coordinates

**Entrada:** Nombre de un archivo (ruta)
**Salida :** Lista de coordenadas no repetidas

```
1 while Archivo(nombre dado) esté abierto do
2     foreach renglones ← documento do
3         if (latitud , longitud ) no están en la lista then
4             Agreagrlas
```

[1]Número de cuenta 317031326
[2]Número de cuenta n

**Función 2:** read_csv_file

**Entrada:** Nombre de un archivo (ruta)
**Salida :** Lista de diccionarios con vuelos

```
1  try:
2      Abrir ruta
3      for linea ← archivo do
4          linea ← lista
5  catch FileNotFoundError:
6      muesta Error, escribe una ruta válida
7      exit
8  catch FileExistsError:
9      muesta Error, archivo válido
10     exit
```

**Función 3:** read_headers

**Entrada:** Nombre de un archivo (ruta)
**Salida :** Lista de cabeceras

```
1  with:
2      lector ← Leer primera linea( ruta)
```

## main.py

**Función 4:** validate_file

**Entrada:** Nombre de un archivo (ruta) pasados como argumento al programa

```
if longiutd del argumento no es 2 then
    muestra: Error
    Debe indicar la ruta a un archivo csv
    salir
if no coincide la extensión .csv then
    muestra: Error, sólo admito archivos csv
    salir
cabezera ← nombres de listas admitidas
entrada_cabecera ← read_headers(argumento[1])
if longitud(entrada_cabecera) no es igual a longitud(cabezera) then
    muestra: ERROR El archivo csv debe tener los siguientes encabezados: origin, destination, origin_latitude,
     origin_longitude, destination_latitude, destination_longitude
    salir
foreach cabeza in entrada_cabezera do
    if cabeza no está en cabezera then
        muestra: ERROR El archivo csv debe tener los siguientes encabezados: origin, destination,
         origin_latitude, origin_longitude, destination_latitude, destination_longitude
        salir
```

**Función 5:** run

**Entrada:** Nombre de un archivo (ruta)

```
validate_file(argumentos al correr el programa);
entradas ← read_csv_file(argumento al correr el programa)
solicitudes_no_repetidas ← read_no_repeated_coordinates(argumentos al iniciar)
foreach solicitud en solicitudes_no_repetidas do
    peticion ← make_api_request_by_coordinates(solicitud[0], solicitud[1])
    peticiones ← setdefault(solicitud, peticion)
foreach entrada en entradas do
    muesta: Datos del clima ;) con formato bonito
```

**Weather.py**

**Función 6:** make_api_request_by _coordinates

**Entrada:** latitud, lontigud
**Salida :** llamada a función parse_weather_info

```
if contador > 59 then
    contador ← 0
    esperar 1 minuto para continuar
get(url + latitud y longitud dadas)
contador ← contador + 1
```

**Función 7:** formato_de_horas

**Entrada:** Número de fecha y hora (unix)
**Salida :** cadena de texto con hora en formato 12 hrs

```
convierte_fotante: numero dado
local_timezone ← get_localzone()()
local_time ← fromtimestamp(flotante, local_timezone)
regresar: local_time con formato de 12 horas, (CODIGO DEL TIEMPO)
```

**Función 8:** parse_weather_info

**Entrada:** respuesta en formato json
**Salida :** llamada a función parse_weather_info

```
try :
    extraer información del archivo json con las llaves proporcionadas por la documentación de la API
catch KeyError:
    regresar: Error, no se pudo consultar la información
regresar: El pronóstico del clima es: X , humedad: x
          Temperatura actual: X°C, mínima: X°C, máxima: X°C Amanecer: X Puesta del sol: X
```

# 5. Mejora a futuro