# Instituto Tecnológico de Costa Rica

# IC4302 - Bases de Datos II

# Documentación Proyecto Opcional

**Profesor**: Nereo Campos Araya

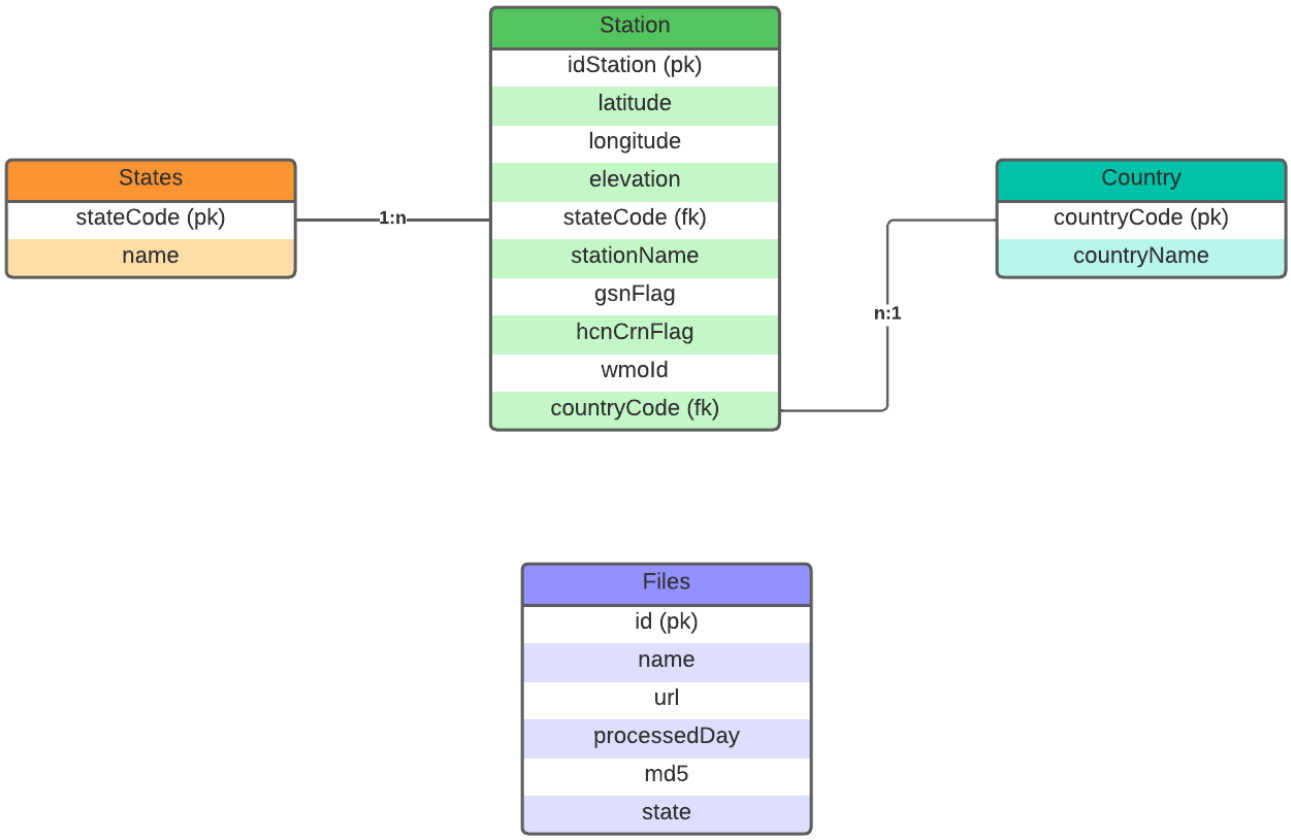
**Estudiantes**:

- Fiorella Zelaya Coto - 2021453615
- Isaac Araya Solano - 2018151703
- Melany Salas Fernández - 2021121147
- Moisés Solano Espinoza - 2021144322

---

## Diagramas

Diagrama Entidad Relación

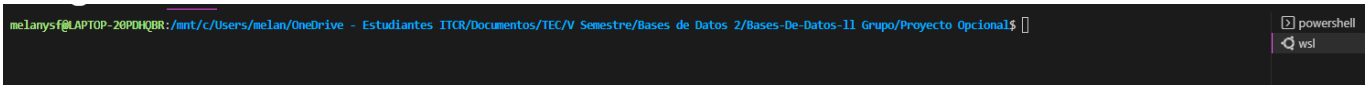



Diagramas en LucidChart

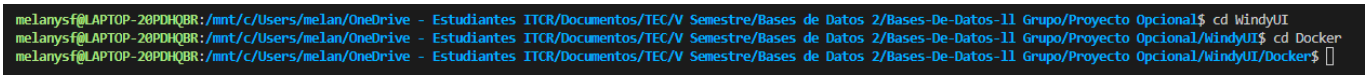
---

## Pasos para la ejecución

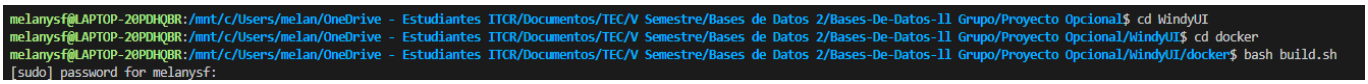
1- Abrir una consola WSL, en la carpeta del proyecto



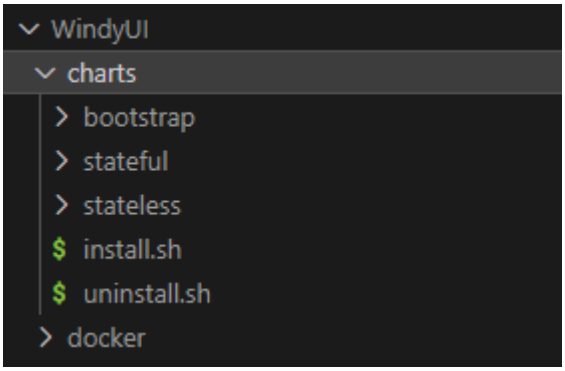

2- Ir a la carpeta "WindyUI" y luego a la carpeta "Docker"



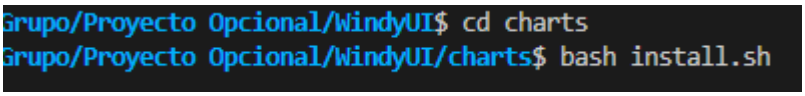

3- Ejecutar el archivo build.sh con el siguiente comando: **bash build.sh**




4- En la misma consola WSL, en la carpeta del proyecto ir a la carpeta "WindyUI" y luego a la carpeta "Charts"




5- Ejecutar el archivo install.sh con el siguiente comando: **bash install.sh**




---

# Componentes

## Countries/States CronJob

El countries y states cronjobs consiste en dos componentes de tipo Cronjob que se ejecutan una vez al día. Estos estan compuestos por una **carpeta app**, ubicada en **WindyUI -> Docker -> CountriesCronjob**, o en caso de states, **StatesCronjob** que contiene:

- **app.py:** Consiste en un achivo python que se encarga de leer el archivo "ghcnd-countries.txt" de la página del NOA, mediante los modulos requests, ademas, se calcula el MD5 del archivo y se crea la conexión con MariaDB, cargando los países a la base de datos de WindyUI.

```python
#-----------------------------------------------------------------------------------------------
# reads countries.txt and inserts in table weather.countries
# @restrictions: none
# @param: none
# @output: noneexecuteProcedure('createCountry', [code, name])
def readCountries():
    url = 'https://www.ncei.noaa.gov/pub/data/ghcn/daily/ghcnd-countries.txt'
    try:
        file = requests.get(url)
    except:
        return "The page is not responding"

    string = file.content.decode('utf-8')
    lines = string.rsplit('\n')

    md5 = getMd5(string)
    stored_results = executeProcedure('loadFile', ["ghcnd-countries.txt", url, str(md5).encode(), "Descargado"])
    print(stored_results)

    for result in stored_results:
        if result[0][0] == "The file has been created" or result[0][0] == 'The textFile has been successfully modified.':
            for line in lines:
                code = line[:2]
                name = line[3:]
                executeProcedure('createCountry', [code, name])
            print("El archivo se modifico")
        else:
            print("El archivo no se modifico")
```

- **requirements.txt:** Archivo que contiene los modulos necesarios para el .py.

```
requests==2.28.2
mysql-connector==2.2.9
```

En el countries/states cronjob tambien se encuentra el **dockerfile** necesario para la creación de la imagen que será usada en el los objetos de cronjobs respectivos.

```
FROM python:3.9

WORKDIR /app

COPY app/. .
RUN pip install --no-cache-dir -r requirements.txt

CMD [ "python", "-u", "./app.py" ]
```

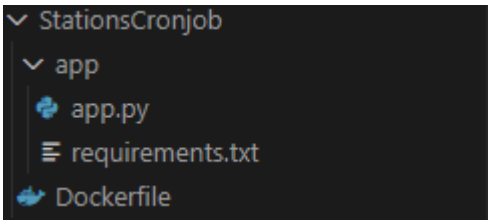
Tambien usan el template **countriesCronjob.yaml** (en caso de countries) o el template **statesCronjob.yaml** (en caso de states), estos estam ubicados en **WindyUI -> charts -> stateless -> template**, cada uno de estos archivos sigue la estructura de un cronjob.

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: {{ .Values.config.countriesCronjob.name }}
spec:
  schedule: "0 5 * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
          - name: {{ .Values.config.countriesCronjob.name }}
            image: {{ .Values.config.countriesCronjob.image }}
            command: ["/bin/sh"]
          restartPolicy: OnFailure
```

Todo esto se conecta con la base de datos en **MariaDB** mediante el modulo **mysql.connector**, que se encarga de hacer una llamada a la funcion **"executeProcedure"**, que recibe el nombre del procedimiento a utilizar, más los parametros del procedimiento, en forma de lista de strings no con None en caso de tener datos NULL.

### Station CronJob

Es similar al anterior, consiste en un componente de tipo Cronjob que se ejecuta una vez al día.




Estos estan compuestos por una **caperta app**, ubicada en **WindyUI -> Docker -> StationCronjob** que contiene:

- **app.py:** Consiste en un achivo python que se encarga de leer el archivo "ghcnd-stations.txt" de la página del NOA, mediante los modulos requests, ademas, se calcula el MD5 del archivo y se crea la conexión con MariaDB, cargando las estaciones a la base de datos de WindyUI.

```python
def readStations():
    url = 'https://www.ncei.noaa.gov/pub/data/ghcn/daily/ghcnd-stations.txt'
    try:
        file = requests.get(url)
    except:
        return "The page is not responding"

    string = file.content.decode('utf-8')
    lines = string.rsplit('\n')

    md5 = getMd5(string)
    stored_results = executeProcedure('loadFile', ["ghcnd-stations.txt", url, str(md5).encode(), "Descargado"])
    print(stored_results)

    for result in stored_results:
        if result[0][0] == "The file has been created" or result[0][0] == 'The textFile has been successfully modified.':
            for line in lines:
                stationId = line[:11]
                countryCode = stationId[0:2]
                latitude = line[12:20].replace(' ', '')
                longitude = line[21:30].replace(' ', '')
                elevation = line[31:37].replace(' ', '')
                state = line[38:40].replace(' ', '')
                name = line[41:71]
                gsnFlag = line[72:75].replace(' ', '')
                hcnFlag = line[76:79].replace(' ', '')
                wmoId = line[80:85].replace(' ', '')

                executeProcedure('createStation', [stationId, latitude, longitude, elevation, state, name, gsnFlag, hcnFlag, wmoId, countryCode])
        else:
            print("El archivo no se modifico")

    file.close()
```

- **requirements.txt:** Archivo que contiene los modulos necesarios para el .py.

```
requests==2.28.2
mysql-connector==2.2.9
```

En el countries/states cronjob tambien se encuentra el **dockerfile** necesario para la creación de la imagen que será usada en el los objetos de cronjobs respectivos.

```
FROM python:3.9

WORKDIR /app

COPY app/. .
RUN pip install --no-cache-dir -r requirements.txt

CMD [ "python", "-u", "./app.py" ]
```

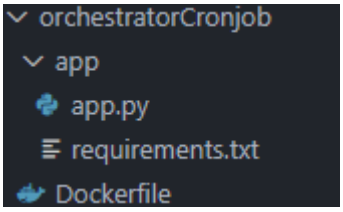
Tambien usan el template **countriesCronjob.yaml** (en caso de countries) o el template **statesCronjob.yaml** (en caso de states), estos están ubicados en **WindyUI -> charts -> stateless -> template**, cada uno de estos archivos sigue la estructura de un cronjob.

De igual forma, se conecta con la base de datos en **MariaDB** mediante el modulo **mysql.connector**, que se encarga de hacer una llamada a la funcion **"executeProcedure"**, que recibe el nombre del procedimiento a utilizar, más los parametros del procedimiento, en forma de lista de strings no con None en caso de tener datos NULL.

## Orchestrator CronJob

Este es un componente tipo Cronjob, que se ejecuta una vez al día y se encarga de listar los archivos de la página **www.ncei.noaa.gov**.

Está estan compuestos por una **caperta app**, ubicada en **WindyUI/Docker/orchestratorCronjob** que contiene:




- **app.py:** este es un archivo Python que se conecta a la página del noaa y lista todos los archivos, se toma cada dirección que se encuentra y se manda un mensaje a la cola de rabbitmq llamada TO_PROCESS, y se agrega o actualiza la tabla weather.files con los datos del archivo. Todo lo que se utiliza en este .py se maneja por variables de entorno, como el nombre de la cola y los credenciales de rabbitmq.

```python
def readFolder():
    print('empezó el request')
    # root of the NOOA website to add to the files url
    root = 'https://www.ncei.noaa.gov/pub/data/ghcn/daily/all/'

    # Making a GET request
    r = requests.get('https://www.ncei.noaa.gov/pub/data/ghcn/daily/all/')

    # Parsing the HTML
    soup = BeautifulSoup(r.content, 'html.parser')

    print('termino el request')

    # find all the anchor tags with "href"
    # the count is used to ommit the first 5 links
    count = 0
    for link in soup.find_all('a', href=True):
        if count > 4:
            name = link['href']
            url = root + link['href']
            state = 'LISTED'
            md5 = None

            # code to save the record and publish the ms in rabbitMQ here
            print(url)

            # message for rabbitmq
            msg = url
            channel.basic_publish(exchange='', routing_key=OUTPUT_QUEUE, body=msg)

            executeProcedure("loadFileFolder", [name, url, md5, state])

        if count == 40:
            break
        count += 1
    r.close()
    # close the rabbitmq connection
    #connection.close()

readFolder()
connection.close()      moisose, last week • folder called WindyUI with application
```

- **requirements.txt:** Archivo que contiene los modulos necesarios para el .py.

```
pika==1.3.1
requests==2.28.2
beautifulsoup4==4.11.2
```

Se usa el template llamado **orchestratorCronjob.yaml** que es de tipo cronjob, ubicado en la carpeta **WindyUI/charts/stateless/templates/**.

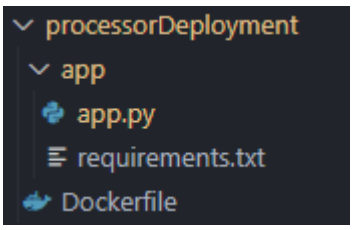
En la carpeta orchestratorCronjob se encuentra el dockerfile para la creación e la imagen y publicación a dockerhub, que será usada en los componentes cronjob.

```
FROM python:3.9

WORKDIR /app

COPY app/. .
RUN pip install --no-cache-dir -r requirements.txt

CMD [ "python", "-u", "./app.py" ]
```

## Processor

Este es un componente de Kubernets tipo deployment que va a estar esperando trabajo de la cola TO_PROCESS de rabbitmq. Lo que hace es tomar el mensaje de la cola y descargar el archivo, calcula el MD5 y se verifica por si existen datos nuevos. Se maneja un índice en Elasticsearch que guardará el nombre y el contenido de los nuevos archivos. También se tendrá otra cola de rabbitmq llamada TO_PROCESS en la que se enviarán mensajes al componente Parser.

Está compuesto por una **caperta app**, ubicada en **WindyUI/Docker/processorDeployment/** que contiene:




- **app.py:** este es un archivo Python que va a recibir mensajes de la cola TO_PROCESS, en cada uno va a venir el enlace de descarga de un archivo. Se calcula el MD5 para saber si es igual o diferente, si es diferente se sube al índice de Elasticsearch y se actualiza el estado en la base de datos de mariadb.
    - Finalmente se publica a la cola TO_PARSE el nombre del archivo para que este sea parseado.

```python
# # reads a single file of the gov/pub/data/ghcn/daily/all/ folder
# # procedure sameFolderFileMD5 is called, if result is 0 the
# # index is added to elastic search
# # @restrictions: none
# # @param: url
# # @output: none
def readFolderFile(url, OUTPUT_QUEUE):
    file = requests.get(url)
    # gets name from url
    fileName = str(url[50:])
    # parse the annoying caracters
    fileName = fileName[2:-1]

    string = str(file.content.decode('utf-8'))

    # changes the md5 if its different, changes the state either way and gets the flag "sameMD5"
    sameMD5 = sameFolderFileMD5("sameFolderFileMD5", [fileName, getMD5(string)+"1", 0])
    print(sameMD5)

    # if the file has changed it is published in elastic search
    if (not sameMD5):

        # elastic search index publication here
        # ** change this for isaac's function
        jsonFile = '{"fileName":"' + fileName + '", ' + '"contents"'+ ':"' + string + '"}'
        jsonF = json.loads({"filename":fileName})
        jsonF.update({"contents":string})
        es.index(index=ESINDEX, id=hashlib.md5(string).hexdigest(), document=jsonF)

        time.sleep(2)

        # publish message to rabbitMQ queue TO_PARSE
        channel_output.basic_publish(exchange='', routing_key=OUTPUT_QUEUE, body=fileName)


def callback(ch, method, properties, body):
    readFolderFile(body, OUTPUT_QUEUE)

credentials_input = pika.PlainCredentials('user', RABBIT_MQ_PASSWORD)
parameters_input = pika.ConnectionParameters(host=RABBIT_MQ, credentials=credentials_input)
connection_input = pika.BlockingConnection(parameters_input)
channel_input = connection_input.channel()

channel_input.queue_declare(queue=INPUT_QUEUE)
channel_input.basic_consume(queue=INPUT_QUEUE, on_message_callback=callback, auto_ack=True)
```

- **requirements.txt:** Archivo que contiene los modulos necesarios para el .py.

```
pika
elasticsearch
requests==2.28.2
mysql-connector==2.2.9
```

Se usa el template llamado **orchestratorCronjob.yaml** que es de tipo cronjob, ubicado en la carpeta **WindyUI/charts/stateless/templates/**.

En la carpeta orchestratorCronjob se encuentra el dockerfile para la creación e la imagen y publicación a dockerhub, que será usada en los componentes cronjob.

```
FROM python:3.9

WORKDIR /app

COPY app/. .
RUN pip install --no-cache-dir -r requirements.txt

CMD [ "python", "-u", "./app.py" ]
```

## Parser

## Otros

- **Función excecuteProcedure**: Es una función que se encarga de hacer la conexión con la base de datos en MariaDB y ejecutar el proceso almacenado que recibe por parametros. Retorna un arreglo con los resultados que da la base de datos al cargar los datos. Si falla en hacer la conexion, retorna un arreglo con el string 'error'.

```
#----------------------------------------Functions----------------------------------
# executes a stored procedure
# @restrictions: none
# @param: the name of the stored prcedure and the parameters of the stored procedure (array of strings)
# @output: none
def executeProcedure(procedure, parameters):
    resultArray = []
    try:
        conn = mysql.connector.connect(host="localhost", user='root', password= pw, port= puerto, database='weather')
        cursor = conn.cursor()
        args = ("FF", 2, 2, 20, 3)
        result_args = cursor.callproc(procedure, parameters)


        for result in cursor.stored_results():
            resultArray.append(result.fetchall())
            #print(resultArray)
        conn.commit()

        if (conn.is_connected()):
            cursor.close()
            conn.close()
            stored_results = cursor.stored_results()
            #print("MySQL connection is closed")


    except mysql.connector.Error as error:
        pass
        #print("Failed to execute stored procedure: {}".format(error))

    finally:
        pass

    return resultArray
```

- **Función getMD5**: Esta función se encarga de calcular el md5 de un archivo, este es usado para saber si el archivo ha sido modificado. Recibe un string con lo que contiene el archivo y retorna el calculo del MD5.

```python
#----------------------------------------
# Calculate the MD5 of a string
# @restrictions: none
# @param: a string
# @output: the hash of the string
def getMd5(string):
    hashSha = hashlib.sha256()
    hashSha.update(string.encode())
    return hashSha.hexdigest()
```

- **Procedure loadFile:** Este procedure se encarga de verificar el md5 de los archivos, para saber si el archivo es totalmente nuevo y hay que crear un nuevo textFile, o si el md5 del archivo cambio y hay que actualizar los datos, o si el archivo sigue igual y no hay cambios en los datos.

```sql
/*
_____________________________________________________________________________________
-> Procedure for update and create files
-> @restrictions: Values null on file name, url or status
-> @param: file name, url, md5 and status
-> @output: result
*/
DELIMITER $$
CREATE PROCEDURE loadFile (fileNameVar VARCHAR(50), urlVar VARCHAR(100), fileMd5Var VARCHAR(130),
                           fileStatusVar VARCHAR(20))
BEGIN
    IF ISNULL(fileNameVar) OR ISNULL(urlVar) OR ISNULL(fileMd5Var) OR ISNULL(fileStatusVar) THEN
        SELECT "There are values NULL";
    ELSEIF (SELECT COUNT(*) FROM textFile WHERE fileName = fileNameVar) = 0 THEN
        CALL createTextFile(fileNameVar, urlVar, fileMd5Var, fileStatusVar);
        SELECT "The file has been created";
    ELSEIF (SELECT fileMd5 FROM textFile WHERE fileName = fileNameVar) = fileMd5Var THEN
        SELECT "The file has no changes";
    ELSE
        CALL updateTextFile(fileNameVar, NULL, (SELECT DATE(NOW())), fileMd5Var, fileStatusVar);
    END IF;
END;
$$
```

- **Procedure readCountries:** Este procedure se encarga de descargar el archivo txt que contiene los datos (codigo de pais, nombre de pais) mediante la utilizacion de la libreria requests. Una vez descargado, calcula el md5 utilizando la funcion getMd5(). Luego, mediante la funcion executeProcedure() se ejecuta el proceso almacenado loadFile en MariaDB para revisar el md5 y guardar el archivo en la base de datos *weather*. Finalmente, si la conexion con la base de datos falla se retorna un string 'Conexion fallida', de lo contrario, dependiendo del resultado del proceso almacenado, se lee el archivo txt y se agregan los datos a la base de datos o simplemente no se modifica el archivo. Se retorna un string "El archivo se modifico" o "El archivo no se modifico".

```python
#-----------------------------------------------------------------------------------------------
# reads countries.txt and inserts in table weather.countries
# @restrictions: none
# @param: none
# @output: noneexecuteProcedure('createCountry', [code, name])
def readCountries():
    url = 'https://www.ncei.noaa.gov/pub/data/ghcn/daily/ghcnd-countries.txt'
    try:
        file = requests.get(url)
    except:
        return "The page is not responding"

    string = file.content.decode('utf-8')
    lines = string.rsplit('\n')

    md5 = getMd5(string)
    stored_results = executeProcedure('loadFile', ["ghcnd-countries.txt", url, str(md5).encode(), "Descargado"])
    print(stored_results)

    for result in stored_results:
        if result[0][0] == "The file has been created" or result[0][0] == 'The textFile has been successfully modified.':
            for line in lines:
                code = line[:2]
                name = line[3:]
                executeProcedure('createCountry', [code, name])
            print("El archivo se modifico")
        else:
            print("El archivo no se modifico")
```

- **Procedure readStates:** Este procedure se encarga de descargar el archivo txt que contiene los datos (codigo de estado, nombre de estado) mediante la utilizacion de la libreria requests. Una vez descargado, calcula el md5 utilizando la funcion getMd5(). Luego, mediante la funcion executeProcedure() se ejecuta el proceso almacenado loadFile en MariaDB para revisar el md5 y guardar el archivo en la base de datos *weather*. Finalmente, si la conexion con la base de datos falla se retorna un string 'Conexion fallida', de lo contrario, dependiendo del resultado del proceso almacenado, se lee el archivo txt y se agregan los datos a la base de datos o simplemente no se modifica el archivo. Se retorna un string "El archivo se modifico" o "El archivo no se modifico".

```python
#-------------------------------------------------------------------------------------------------------------------
# reads states.txt and inserts in table weather.states
# @restrictions: none
# @param: none
# @output: none
def readStates():
    varResult = ''
    url = 'https://www.ncei.noaa.gov/pub/data/ghcn/daily/ghcnd-states.txt'
    try:
        file = requests.get(url)
    except:
        return "The page is not responding"

    string = file.content.decode('utf-8')
    lines = string.rsplit('\n')

    md5 = getMd5(string)
    stored_results = executeProcedure('loadFile', ["ghcnd-states.txt", url, str(md5).encode(), "Descargado"])
    print(stored_results)

    if stored_results[0] == 'error':
        return 'Conexion fallida'

    for result in stored_results:
        if result[0][0] == "The file has been created" or result[0][0] == 'The textFile has been successfully modified.':
            for line in lines:
                code = line[:2]
                name = line[3:]
                executeProcedure('createState', [code, name])
            varResult = 'El archivo se modifico'
        else:
            print("El archivo no se modifico")
            varResult = "El archivo no se modifico"

    file.close()
    return varResult


#_____________________________________________MAIN_____________________________________________________________
readStates()
```

- **Procedure readStations:** Este procedure se encarga de descargar el archivo txt que contiene los datos (id, codigo de pais, latitud, longitud, elevacion, estado, nombre, gsn, hcn, wmo) mediante la utilizacion de la libreria requests. Una vez descargado, calcula el md5 utilizando la funcion getMd5(). Luego, mediante la funcion executeProcedure() se ejecuta el proceso almacenado loadFile en MariaDB para revisar el md5 y guardar el archivo en la base de datos *weather*. Finalmente, si la conexion con la base de datos falla se retorna un string 'Conexion fallida', de lo contrario, dependiendo del resultado del proceso almacenado, se lee el archivo txt y se agregan los datos a la base de datos o simplemente no se modifica el archivo. Se retorna un string "El archivo se modifico" o "El archivo no se modifico".

```python
def readStations():
    varResult = ''
    url = 'https://www.ncei.noaa.gov/pub/data/ghcn/daily/ghcnd-stations.txt'
    try:
        file = requests.get(url)
    except:
        return "The page is not responding"

    string = file.content.decode('utf-8')
    lines = string.rsplit('\n')

    md5 = getMd5(string)
    stored_results = executeProcedure('loadFile', ["ghcnd-stations.txt", url, str(md5).encode
    print(stored_results)

    if stored_results[0] == 'error':
        return 'Conexion fallida'

    for result in stored_results:
        if result[0][0] == "The file has been created" or result[0][0] == 'The textFile has b
            for line in lines:
                stationId = line[:11]
                countryCode = stationId[0:2]
                latitude = line[12:20].replace(' ', '')
                longitude = line[21:30].replace(' ', '')
                elevation = line[31:37].replace(' ', '')
                state = line[38:40].replace(' ', '')
                name = line[41:71]
                gsnFlag = line[72:75].replace(' ', '')
                hcnFlag = line[76:79].replace(' ', '')
                wmoId = line[80:85].replace(' ', '')

                executeProcedure('createStation', [stationId, latitude, longitude, elevation,
            varResult = 'El archivo se modifico'
        else:
            print("El archivo no se modifico")
            varResult = 'El archivo no se modifico'

    file.close()
    return varResult
```

- **Función processorJson** Esta es una función que se encarga de recibir la información de los archivos publicados en la cola to_process y transformaro al formato Json necesario, donde se colocan tanto el nombre del archivo como sus contenidos.

```python
"""
This method receives the name of a file and its contents and creates a file called processor.json
"""
def processorJson(name, contents):
    result = {"filename" : name, "contents": contents}

    with open("processor.json", "w") as write_file:
        json.dump(result, write_file)

    return result
```

- **Función parserJson** Esta es una función que se encarga de transformar la información de cada set de datos extraídos del archivo y convertirlos en formato Json para luego ser publicados en la cola to_transform. En esta se recibe la información del station_id, date, type, value, mflag, qflag y sflag. Después de recibir esa información se crea un solo json que tiene el nombre del archivo y luego la lista de datos.

```
"""
This method receives a filename and a list of data and creates a file called parser.json
"""

def parserJson(filename, listOfData):

    dictionaryResults = []
    for i in listOfData:
        tempDict = {
            "station_id": i[0],
            "date": i[1],
            "type": i[2],
            "value": i[3],
            "mflag": i[4],
            "qflag": i[5],
            "sflag": i[6]
        }
        dictionaryResults.append(tempDict)

    result = {"filename" : filename,
              "data" : dictionaryResults}

    with open("parser.json", "w") as write_file:
        json.dump(result, write_file)

    return result
```

- **Función transformationJson** Esta es una función que recibe un archivo json proveniente de la cola to_transform que tiene el formato producido por la función parserJson. El objetivo de esta función es obtener más información a partir del json recibido fragmentando la fecha, el station_id y agregando el nombre del type. En este método se utiliza un diccionario que incluye los tipos indicados en la especificación del proyecto. De ese diccionario se extrae el nombre completo que se relaciona al acrónimo presente en el atributo type de cada json. Además se extrae de la fecha el mes y el año de los datos y luego se extrae del station_id el código de país, el código de network y el idetificador de estación real.

Este es el diccionario de tipos:

```
typeNames = {
    "PRCP" : "Precipitation (tenths of mm)",
    "SNOW" : "Snowfall (mm)",
    "SNWD" : "Snow depth (mm)",
    "TMAX" : "Maximum temperature (tenths of degrees C)",
    "TMIN" : "Minimum temperature (tenths of degrees C)",
    "RHMX" : "Maximum relative humidity for the day (percent)"
}
```

Función transformationJson:

```
This method receives a json and transform its data and creates a file called transformation.json
"""
def transformationJson(jsonParsed):
    for i in json["data"]:
        date = i["date"]
        stationId = i["station_id"]

        tempDict = {
            "station_id": i["station_id"],
            "date": i["date"],
            "month": date[0:4],
            "year": date[4:6],
            "type": i["type"],
            "value": i["value"],
            "mflag": i["mflag"],
            "qflag": i["qflag"],
            "sflag": i["sflag"],
            "FIPS_country_code" : stationId[0:2],
            "network_code" : stationId[2:3],
            "real_station_id" : stationId[3:12],
            "type_name" : typeNames[i["type"]]
        }

        i = tempDict

    with open("transformation.json", "w") as write_file:
        json.dump(jsonParsed, write_file)

    return jsonParsed
```

---

# Pruebas

## Prueba de Station CronJob

Es esta prueba se verifica que la conexión de la base de datos se haga correctamente.

```
import unittest

from app import *

class test_mariadb(unittest.TestCase):

    def test_createStations(self):
        result = readStations()
        self.assertNotEqual(result, "Conexion fallida")

if __name__=="__main__":
    unittest.main()
```

## Prueba de Countries/States CronJob

Es esta prueba se verifica que la conexión de la base de datos se haga correctamente.

```python
import unittest

from app import *

class test_mariadb(unittest.TestCase):

    def test_createStates(self):
        result = readStates()
        self.assertNotEqual(result, "Conexion fallida")

if __name__=="__main__":
    unittest.main()
```

### Prueba de Orchestrator CronJob

```python
import unittest

from app import *

class test_mariadb(unittest.TestCase):

    def test_createCountry(self):
        result = readFolder()
        self.assertNotEqual(result, "Conexion fallida")

if __name__=="__main__":
    unittest.main()
```

### Prueba de Processor

### Prueba de Parser

---

## Resultados de pruebas unitarias

### Prueba de Station CronJob

```
------
 > [6/6] RUN python test_mariadb.py:
#10 22.49 F
#10 22.49 ======================================================================
#10 22.49 FAIL: test_createStations (__main__.test_mariadb)
#10 22.49 ----------------------------------------------------------------------
#10 22.49 Traceback (most recent call last):
#10 22.49   File "/app/test_mariadb.py", line 9, in test_createStations
#10 22.49     self.assertNotEqual(result, "Conexion fallida")
#10 22.49 AssertionError: 'Conexion fallida' == 'Conexion fallida'
#10 22.49
#10 22.49 ----------------------------------------------------------------------
#10 22.49 ['error']
#10 22.49 ['error']
#10 22.49 Ran 1 test in 8.204s
#10 22.49
#10 22.49 FAILED (failures=1)
------
executor failed running [/bin/sh -c python test_mariadb.py]: exit code: 1
```

## Prueba de Countries/States CronJob

```
#11 2.585     self.assertNotEqual(result, "Conexion fallida")
#11 2.585 AssertionError: 'Conexion fallida' == 'Conexion fallida'
#11 2.585
#11 2.585 ----------------------------------------------------------------------
#11 2.585 Ran 1 test in 0.746s
#11 2.585
#11 2.585 FAILED (failures=1)
------
executor failed running [/bin/sh -c python test_mariadb.py]: exit code: 1
```

### States

Cuando la conexión con la base datos falla, la imagen no se crea.

```
 > [6/6] RUN python test_mariadb.py:
#11 2.621 /app/app.py:40: ResourceWarning: unclosed <socket.socket fd=3, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=6, laddr=('0.0.0.0', 40382)>
#11 2.621   return ['error']
#11 2.621 ResourceWarning: Enable tracemalloc to get the object allocation traceback
#11 2.621 F
#11 2.622 ======================================================================
#11 2.622 FAIL: test_createStates (__main__.test_mariadb)
#11 2.622 ----------------------------------------------------------------------
#11 2.622 Traceback (most recent call last):
#11 2.622   File "/app/test_mariadb.py", line 9, in test_createStates
#11 2.622     self.assertNotEqual(result, "Conexion fallida")
#11 2.622 AssertionError: 'Conexion fallida' == 'Conexion fallida'
#11 2.622
#11 2.622 ----------------------------------------------------------------------
#11 2.623 Ran 1 test in 0.452s
#11 2.623
#11 2.623 FAILED (failures=1)
#11 2.623 ['error']
#11 2.623 ['error']
```

## Prueba de Orchestrator CronJob

```
------
 > [6/6] RUN python test_orchestrator.py:
#9 0.817 Traceback (most recent call last):
#9 0.817   File "/app/test_orchestrator.py", line 3, in <module>
#9 0.817     from app import *
#9 0.817   File "/app/app.py", line 27, in <module>
#9 0.817     parameters = pika.ConnectionParameters(host=RABBIT_MQ, credentials=credentials)
#9 0.817   File "/usr/local/lib/python3.10/site-packages/pika/connection.py", line 628, in __init__
#9 0.817     self.host = host
#9 0.817   File "/usr/local/lib/python3.10/site-packages/pika/connection.py", line 350, in host
#9 0.817     validators.require_string(value, 'host')
#9 0.817   File "/usr/local/lib/python3.10/site-packages/pika/validators.py", line 15, in require_string
#9 0.817     raise TypeError('%s must be a str or unicode str, but got %r' % (
#9 0.817 TypeError: host must be a str or unicode str, but got None
------
executor failed running [/bin/sh -c python test_orchestrator.py]: exit code: 1
```

**Prueba de Processor**

**Prueba de Parser**

---

# Recomendaciones

1- Empezar por lo primero, no apresurarse y empezar con partes del proyecto que estan más avanzadas.
2- Repartir y asignar tareas a cada integrante del equipo para progresar.
3- Investigar los conceptos esenciales para desarrollar la solución.
4- Tener un buen conocimiento de como se utilizan las herramientas necesarias para el desarrollo del proyecto.
5- Utilizar un buen control de versiones y tener un buen manejo de github.
6- Tener una buena estructura del proyecto y dividir el proyecto de forma funcional.
7- Implementar buenas prácticas de programación.
8- Tener una buena comunicación con el equipo de trabajo.
9- Realizar pruebas.
10- Seguir aprendiendo y enriqueciendo el conocimiento después de finalizar el proyecto.

---

# Conclusiones

1- La organización es importante para poder llevar a cabo el proyecto.
2- El trabajo en equipo es esencial para llevar finalizar el proyecto.
3- El conocimiento de conceptos básicos es sustancial para entender los pasos que hay que realizar al desarrollar el proyecto.
4- El tener un buen entendimiento del funcionamiento de las herramientas que se van a necesitar facilita el avance del desarrollo de la solución.
5- El tener un buen control de versiones y saber utilizar github facilita el trabajo en equipo y es una buena práctica.
6- El tener una buena organización y estructura del proyecto es importante para tener un mayor orden, y por ende, facilitar el trabajo.
7- El utilizar buenas prácticas de programación asegura que el código sea legible y entendible para continuar su desarrollo en un futuro de forma eficaz.
8- El tener una buena comunicación tiene como resultado un proyecto de calidad y organizado que avanza

progresivamente.
9- Es importante la realización de pruebas para garantizar el buen funcionamiento del programa.
10- Para reforzar habilidades y mejorar de forma continua, es importante continuar la investigación de los temas que se estudiaron.