Instituto Tecnológico de Costa Rica

IC4302 - Bases de Datos II

Documentación Proyecto Opcional

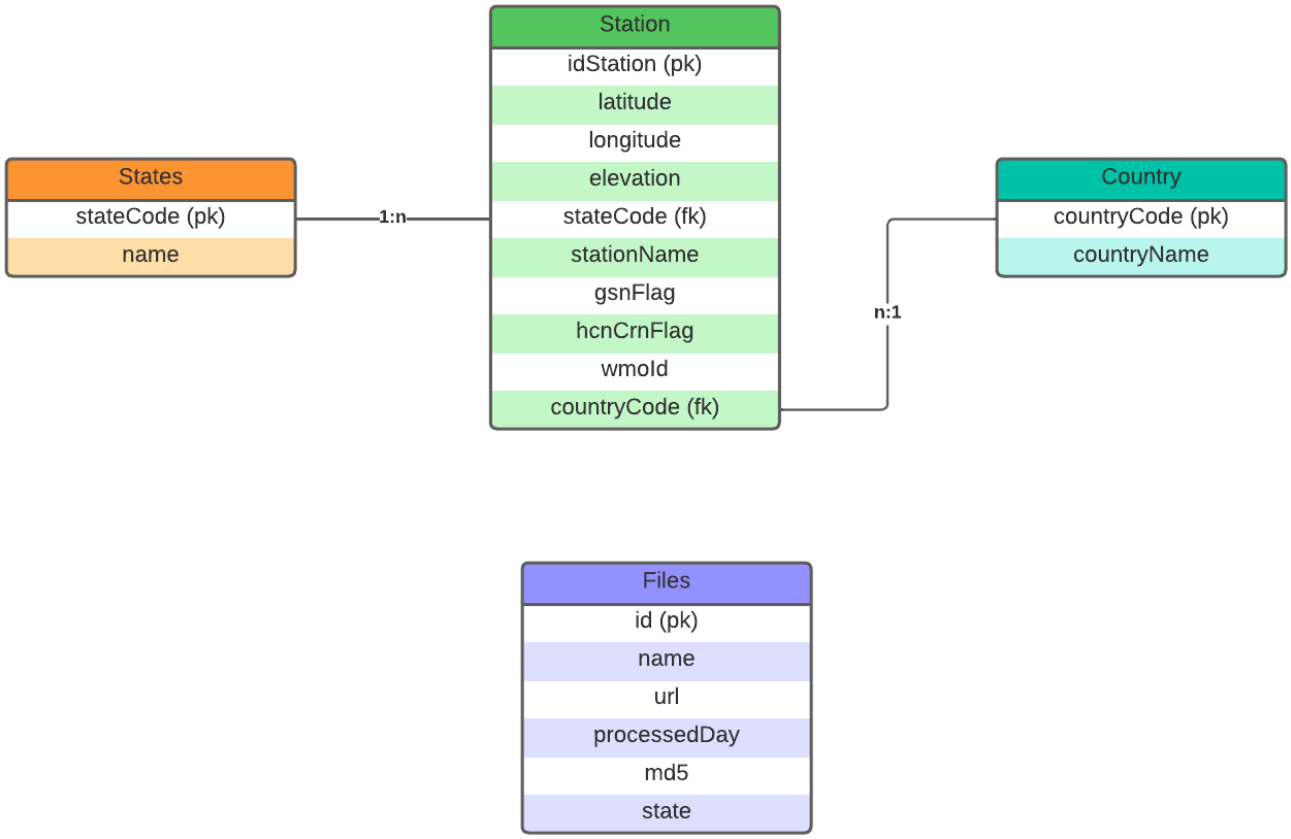
Profesor: Nereo Campos Araya

Estudiantes:

- Fiorella Zelaya Coto - 2021453615
- Joxan Fuertes Villegas -
- Isaac Araya Solano -
- Melany Salas Fernández - 2021121147
- Moisés Solano Espinoza - 2021

Diagramas

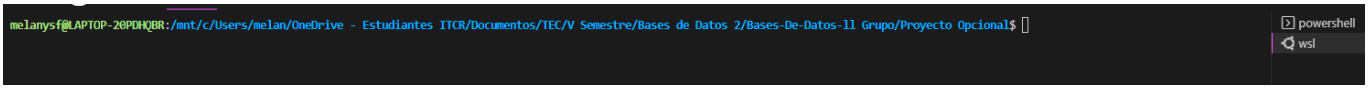
Diagrama Entidad Relación



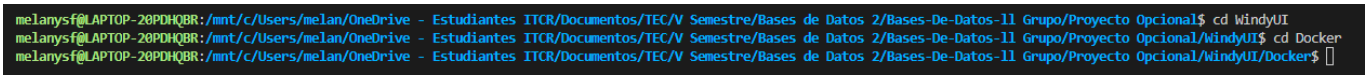
Diagramas en LucidChart

Pasos para la ejecución

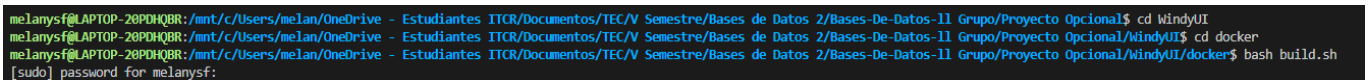
1- Abrir una consola WSL, en la carpeta del proyecto



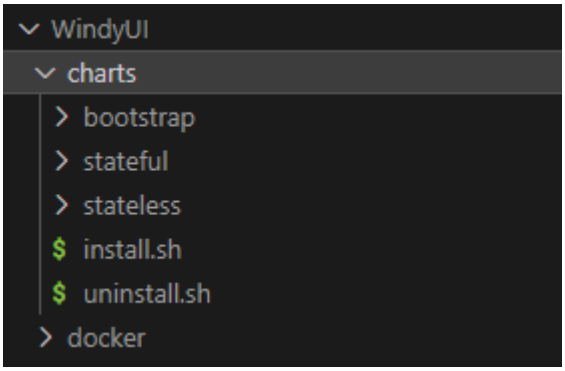
2- Ir a la carpeta "WindyUI" y luego a la carpeta "Docker"



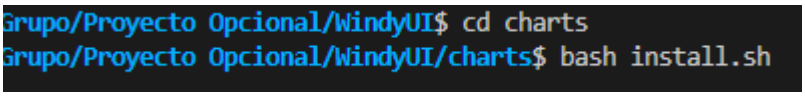
3- Ejecutar el archivo build.sh con el siguiente comando: **bash build.sh**



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 **caperta 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" ]
```

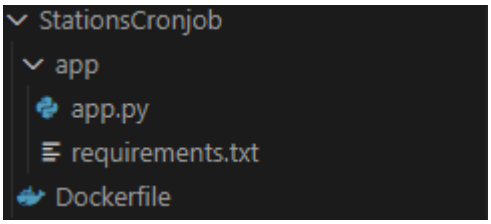
Tambien usan el template **countriesCronjob.yaml** (en caso de countries) o el template **statesCronjob.yaml** (en caso de states), estos estam ubicados en **WindyUI -> charts -> stateless -> template**, cada uno de estos archivos sigue la estructura de un cronjob.

```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 estam 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

Processor

Parser

Otros

- **Función excecuteProcedure**: Es una función que se encarga de hacer la conexión con la base de datos en MariaDB. Retorna un arreglo con los resultados que da la base de datos al cargar los datos.

```
#----------------------------------------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.

```
#----------------------------------------
# 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.

```
/*
_____________________________________________________________________________________
-> 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 $$
CREATE PROCEDURE loadFile (fileNameVar VARCHAR(50), urlVar VARCHAR(100), fileMd5Var VARCHAR(130),
                           fileStatusVar VARCHAR(20))
BEGIN
    IF ISNULL(fileNameVar) OR ISNULL(urlVar) OR ISNULL(fileMd5Var) OR ISNULL(fileStatusVar) THEN
       SELECT "There are values NULL";
    ELSEIF (SELECT COUNT(*) FROM textFile WHERE fileName = fileNameVar) = 0 THEN
       CALL createTextFile(fileNameVar, urlVar, fileMd5Var, fileStatusVar);
       SELECT "The file has been created";
    ELSEIF (SELECT fileMd5 FROM textFile WHERE fileName = fileNameVar) = fileMd5Var THEN
       SELECT "The file has no changes";
    ELSE
       CALL updateTextFile(fileNameVar, NULL, (SELECT DATE(NOW())), fileMd5Var, fileStatusVar);
    END IF;
END;
$$
```

Pruebas

Prueba de Station CronJob

Prueba de Countries/States CronJob

Prueba de Orchestrator CronJob

Pruebo de Processor

Pruebo de Parser

Resultados de pruebas unitarias

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.

Referencias Bibliográficas

(La documentación debe cubrir todos los componentes implementados o instalados/configurados, en caso de que algún componente no se encuentre implementado, no se podrá documentar y tendrá un impacto en la completitud de la documentación.)