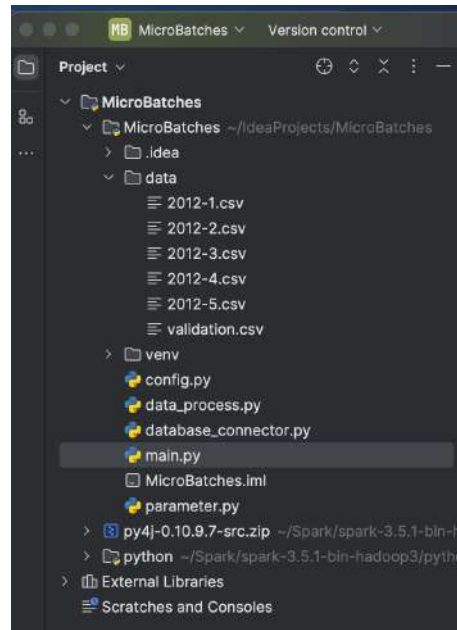


NOTAS - SOLUCIÓN PROPUESTA

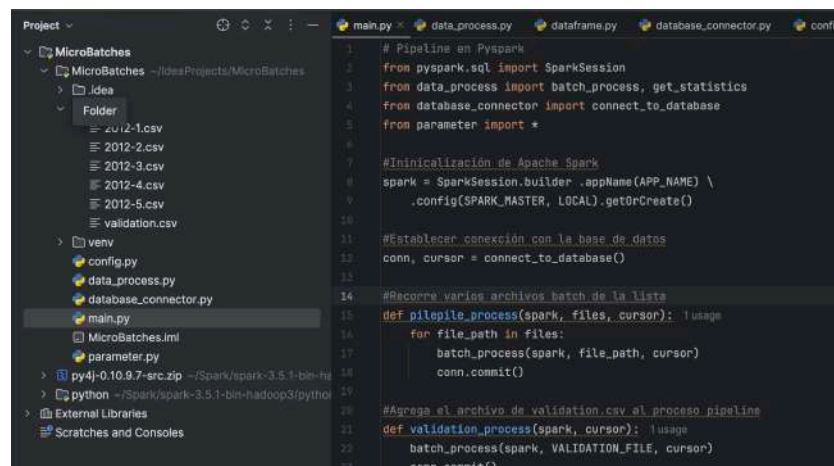
Para la solución del reto se usaron las siguientes tecnologías.

- Lenguaje de programación: Python
- Framework: PySpark
- Bases de datos: PostgreSQL

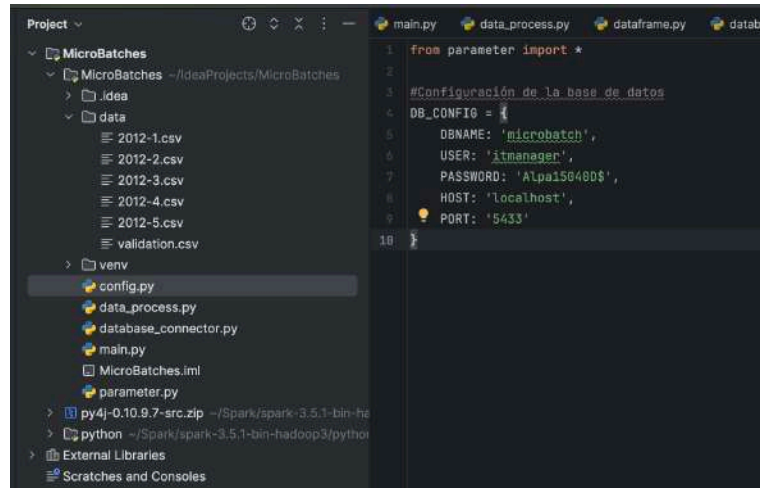
La solución se creó usando un proyecto Python en IntelliJ con la siguiente estructura:



- main.py: Archivo principal del proyecto pipeline



- config.py: Archivo de configuración de la base de datos

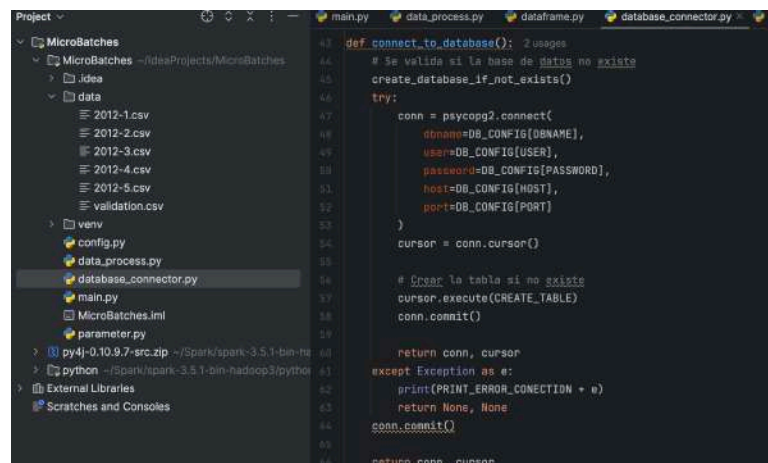


```

1 from parameter import *
2
3 #Configuración de la base de datos
4 DB_CONFIG = {
5     DBNAME: 'microbatch',
6     USER: 'itmanager',
7     PASSWORD: 'Alpa150480$',
8     HOST: 'localhost',
9     PORT: '5433'
10 }

```

- database_connector.py: Archivo que contiene los métodos de conexión a la base de datos PostgreSQL

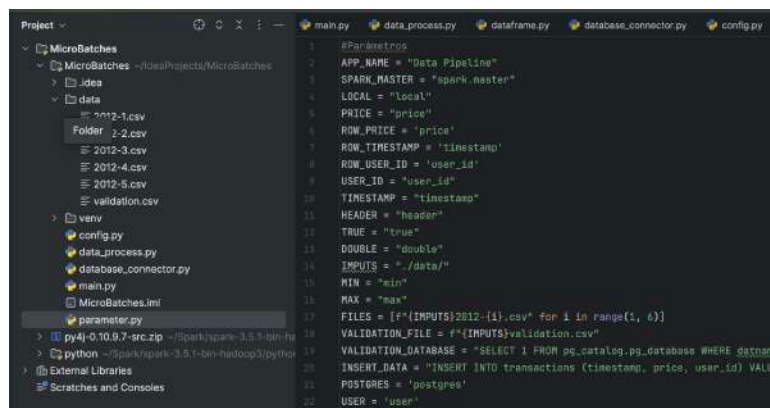


```

43 def connect_to_database(): 2 usages
44     # Se valida si la base de datos no existe
45     create_database_if_not_exists()
46     try:
47         conn = psycopg2.connect(
48             dbname=DB_CONFIG[DBNAME],
49             user=DB_CONFIG[USER],
50             password=DB_CONFIG[PASSWORD],
51             host=DB_CONFIG[HOST],
52             port=DB_CONFIG[PORT]
53         )
54         cursor = conn.cursor()
55
56         # Crear la tabla si no existe
57         cursor.execute(CREATE_TABLE)
58         conn.commit()
59
60         return conn, cursor
61     except Exception as e:
62         print(PRINT_ERROR_CONNECTION + e)
63         return None, None
64     conn.commit()
65
66     return conn, cursor

```

- parameter.py: Archivo con parametrías usadas en el proyecto.



```

1 #Parametros
2 APP_NAME = "Data Pipeline"
3 SPARK_MASTER = "spark.master"
4 LOCAL = "local"
5 PRICE = "price"
6 ROW_PRICE = "price"
7 ROW_TIMESTAMP = "timestamp"
8 ROW_USER_ID = "user_id"
9 USER_ID = "user_id"
10 TIMESTAMP = "timestamp"
11 HEADER = "header"
12 TRUE = "true"
13 DOUBLE = "double"
14 INPUTS = "data/"
15 MIN = "min"
16 MAX = "max"
17 FILES = [f"{INPUTS}2012-{i}.csv" for i in range(1, 6)]
18 VALIDATION_FILE = f"{INPUTS}validation.csv"
19 VALIDATION_DATABASE = "SELECT 1 FROM pg_catalog.pg_database WHERE datname"
20 INSERT_DATA = "INSERT INTO transactions (timestamp, price, user_id) VALUES"
21 POSTGRES = "postgres"
22 USER = "user"

```

- database_connector.py: Archivo que contiene los métodos para el procesamiento de los datos en batch y las estadísticas finales de la base de datos

```

def batch_process(spark, file_path, cursor):
    14 def batch_process(spark, file_path, cursor):
    15     global TOTAL_COUNT, TOTAL_SUM, MIN_PRICE, MAX_PRICE, AVE_PRICE
    16     df_batch = (spark.read.option(HEADER, TRUE).csv(file_path)
    17                 .withColumn(PRICE, col(PRICE).cast(DOUBLE)))
    18
    19     #Cálculo de variables de seguimiento por cada batch
    20     count_batch = df_batch.count()
    21     min_batch = df_batch.select(min(PRICE)).collect()[0][0]
    22     max_batch = df_batch.select(max(PRICE)).collect()[0][0]
    23     sum_batch = df_batch.select(sum(PRICE)).collect()[0][0]
    24
    25     #Insertamos a la base de datos de los datos batch
    26     rows = df_batch.collect()
    27     for row in rows:
    28         cursor.execute(INSERT_DATA,
    29                        (row[ROW_TIMESTAMP], row[ROW_PRICE], row[ROW_USER_ID]))
    30
    31     #Actualización de las variables
    32     TOTAL_COUNT += count_batch
    33     TOTAL_SUM += sum_batch
    34     MIN_PRICE = python_min(MIN_PRICE, min_batch)
    35     MAX_PRICE = python_max(MAX_PRICE, max_batch)
    36     AVE_PRICE = TOTAL_SUM / TOTAL_COUNT

```

Para la ejecución del proyecto se debe contar con una base de datos local de PostgreSQL

Evidencias:

Al ejecutar el pipeline se procesan los datos de acuerdo a las especificaciones del documento, a continuación, se muestra evidencia de las estadísticas de cargue de la información antes y después de cargar el archivo validations.csv con la información de la base de datos.

```

27 get_statistics(cursor)
30
31 #Cierre de Spark y la conexión con la base de datos
32 cursor.close()
33 conn.close()
34 spark.stop()
35
36
37

```

```

/Users/itmanager/IdeaProjects/MicroBatches/venv/bin/python /Users/itmanager/IdeaProjects/MicroBatches/main.py
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/05/28 21:16:34 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
PROCESAMIENTO BATCH ./data/2012-1.csv: Count = 22, Avg = 54.22727272727273, Min = 14.0, Max = 97.0
PROCESAMIENTO BATCH ./data/2012-2.csv: Count = 51, Avg = 54.568627450980394, Min = 10.0, Max = 100.0
PROCESAMIENTO BATCH ./data/2012-3.csv: Count = 82, Avg = 56.5, Min = 10.0, Max = 100.0
PROCESAMIENTO BATCH ./data/2012-4.csv: Count = 112, Avg = 55.714285714285715, Min = 10.0, Max = 100.0
PROCESAMIENTO BATCH ./data/2012-5.csv: Count = 143, Avg = 56.26573426573427, Min = 10.0, Max = 100.0
ESTADISTICAS DE PROCESAMIENTO EN BASE DE DATOS: Count = 143, Avg = 56.26573426573427, Min = 10.0, Max = 100.0
PROCESAMIENTO BATCH ./data/validation.csv: Count = 151, Avg = 55.496688741721854, Min = 10.0, Max = 100.0
ESTADISTICAS DE PROCESAMIENTO EN BASE DE DATOS: Count = 151, Avg = 55.496688741721854, Min = 10.0, Max = 100.0

Process finished with exit code 0

```

- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > 1.3 Sequences
- ✓ Tables (1)
 - ✓ transactions
 - > Columns
 - > Constraints
 - > Indexes
 - > RLS Policies
 - > Rules
 - > Triggers
- > Trigger Functions
- > Types
- ✓ Views
- Subscriptions
- postgres
- | Casts

public.transactions/microbatch/itmanager@SQL

Query Query History

```
1 SELECT * FROM public.transactions
2
```

Data Output Messages Notifications

	timestamp text	price real	user_id integer
1	1/10/2012	50	9
2	1/11/2012	87	10
3	1/12/2012	64	7
4	1/13/2012	20	10
5	1/14/2012	14	10
6	1/15/2012	95	8
7	1/16/2012	95	2
8	1/17/2012	62	4
9	1/18/2012	46	8
10	1/19/2012	97	2