



**UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS**

Universidad Distrital Francisco José de Caldas

Facultad Tecnológica

Hackaton LogView360

David Alexis Lozano Clavijo – 20221578010

Juan Pablo Daza Bejarano- 20221578029

Daniel Felipe Bustamante Pérez - 20221578028

Banco de Occidente

Documento técnico

30 de mayo de 2025

Contenido

Introducción	3
Abstract	3
Desarrollo	3
Diagrama con componentes desplegables	6
Funcionalidad de cada componente	7
Conclusión	8

Introducción

En el presente documento se expone la solución implementada por el grupo para la hackathon LogView 360, cuyo objetivo es visualizar, entender y analizar el recorrido completo de las transacciones bancarias, desde su validación hasta su ejecución final en el Core Bancario.

Abstract

This document presents the solution implemented by the group for the LogView 360 hackathon, whose objective is to visualize, understand and analyze the complete path of banking transactions, from their validation to their final execution in the Core Banking.

Desarrollo

Para presentar una solución al problema plantado, se realizó un análisis inicial del contenido de cada uno de los archivos logs. Para la solución del punto 1 se realizó el diseño del contenido de cada uno de los logs. Para pasar los 3 formatos de logs a una estructura homogénea se optó por usar una base de datos relacional. Se eligió Neon, una plataforma serverless de PostgreSQL en la nube, como sistema de almacenamiento.

Para entender las diferencias y similitudes entre los formatos de los logs se utilizó Excel como herramienta de apoyo para visualizar el contenido de cada archivo y facilitar el análisis previo.

logs_SC.json		
timestamp	timestamp	
transaction_id	varchar	
user_id	varchar	
ip_address	varchar	
resultado_validación	enum	Aprobada o rechazada
Motivo_fallo	varchar	recibe nulos
modulo	varchar	
verificaciones_realizadas	varchar	

log_midFlow		
timestamp	timestamp	
nivel_log	enum	INFO o ERROR
transaction_id	varchar	FK
direction	enum	REQUEST o RESPONSE
operation	enum	TRANSFERIR/RENTAR/TRANSFEREIR
status_code	enum	200 o 500
latency_ms	int	recibe nulos
user_id	varchar	FK
ip_address	varchar	FK
modulo	varchar	

logs_Core_Bank.log		
timestamp	timestamp	
Nivel_log	varchar	
modulo	varchar	
user_id	varchar	
ip_address	varchar	
transaction_id	varchar	
tipo_transaccion	varchar	
tipo_cuenta	enum	ahorros o corriente
estado	varchar	siempre completado
valor	float	

A partir del análisis realizado con Excel, se pudo identificar el tipo de dato adecuado para cada dato del archivo y se detectó la presencia de información repetida entre

ellos.

Como resultado, se definió el uso de dos entidades para la base de datos Aplicación y Transacción.

- La entidad Aplicación representa cada uno de los pasos por los que pasa la transacción bancaria, entre los cuales están el pasar por SecuChek, MidFlow ESB o CoreBank.
- La entidad Transacción hace referencia a la información relacionada con los datos que se usan en cada una de las tres aplicaciones.

Durante el análisis, se concluyó que existe una relación de muchos a muchos (N:M) entre la entidad Aplicación y Transacción dado que:

- Una aplicación puede procesar muchas transacciones
- Una transacción puede estar en varias aplicaciones

Para modelar correctamente los datos de esta relación, se creó una tabla intermedia llamada Aplicación_transacción, la cual permite asociar los datos de una transacción dentro del contexto de una aplicación específica.

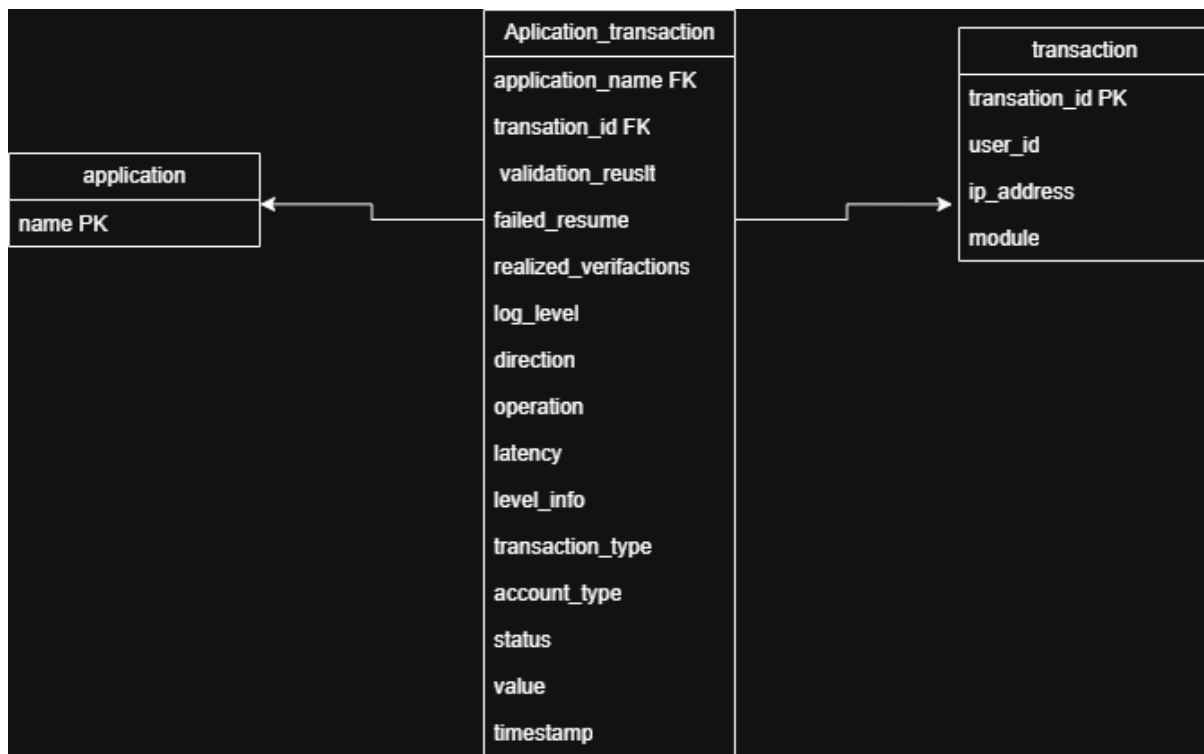
Aplicacion
nombre

enum

Aplicacion_transaccion
resultado_validacion
Motivo_fallo
verificaciones_realizadas
nivel_log
direction
operation
status_code
latency_ms
nivel_info
tipo_transacción
tipo_cuenta
estado
valor
timestamp

Transaccion
trasaction_id
user_id
modulo
ip_address

Para el diseño del modelo relacional (MR) se hizo uso de la herramienta Draw.io



DDL

```
CREATE DATABASE neondb;
```

```
\c neondb;
```

```
CREATE TABLE application(
    name varchar(30) PRIMARY KEY
);
```

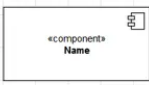







```
CREATE TABLE transaction(
    transaction_id varchar(30) PRIMARY KEY,
    user_id varchar(30) not null,
    module varchar(20) not null,
    ip_address varchar(20) not null
);
```

```
CREATE TABLE applicationtransaction(
```

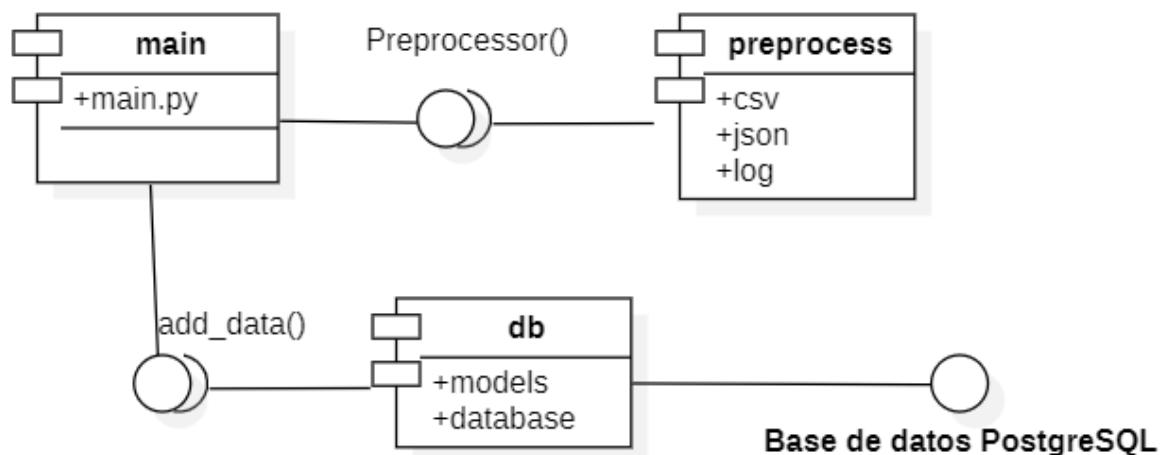
```
transaction_id varchar(30) ,  
  
application_name varchar(30),  
  
state varchar(20),  
  
timestamp timestamp not null,  
  
validate_result varchar(20),  
  
failed_reason varchar(30),  
  
account_type varchar(20),  
  
log_level varchar(20),  
  
operation varchar(20),  
  
direction varchar(20),  
  
status_code integer,  
  
amount real,  
  
transaction_type varchar(20),  
  
realized_verifications varchar(30),  
  
latency varchar(10),  
  
PRIMARY KEY (transaction_id, application_name),  
  
FOREIGN KEY (transaction_id) REFERENCES transaction(transaction_id),  
  
FOREIGN_KEY (application_name) REFERENCES application(name)  
  
);
```

Diagrama con componentes desplegables

Para el diagrama de componentes se usa la siguiente notación

Elemento	Símbolo/notación	Explicación
Componente (<i>component</i>)		Símbolo para representar los módulos de un sistema (la interacción y la comunicación tienen lugar a través de interfaces).
Paquete		Un paquete combina varios elementos del sistema (por ejemplo, clases, componentes o interfaces) en un grupo.
Artefacto		Los artefactos son unidades físicas de información (por ejemplo, código fuente, archivos .exe, scripts o documentos) que se generan en el proceso de desarrollo o el tiempo de ejecución de un sistema o son necesarios para estos.
Interfaz ofrecida		Símbolo para una o más interfaces claramente definidas que proporcionan funciones, servicios o datos al mundo exterior (el semicírculo abierto también se denomina enchufe o <i>socket</i>).
Interfaz requerida		Símbolo de una interfaz necesaria para recibir funciones, servicios o datos del exterior (la notación del círculo con palo también se denomina <i>lollipop</i> o <i>piruleta</i>).
Puerto		Este símbolo indica un punto de interacción independiente entre un componente y su entorno.
Relación		Las líneas actúan como conectores e indican las relaciones entre los componentes.
Relación de dependencia		Conector especial para expresar una relación de dependencia entre los componentes del sistema (no siempre se indica).

A continuación, se presenta el diagrama de la aplicación



Funcionalidad de cada componente

- **Main:** En este módulo se realiza la importación de las funcionalidades definidas en el componente preprocess.

- Preprocess: Este módulo contiene las funcionalidades necesarias para el preprocesamiento de los logs. Se contemplan las funcionalidades para los 3 tipos de archivos CSV, JSON, LOG. Para el manejo de archivos CSV se hace uso de la biblioteca Pandas.
- DB: Este componente contiene los archivos models.py y database.py.
 - En models.py se define el modelo de base de datos relacional utilizando SQLAlchemy.
 - En database.py se configura la conexión a base de datos y se gestionan las operaciones necesarias para su inicialización y acceso.

Conclusión

- La participación en la hackathon nos permitió fortalecer el trabajo en equipo y evaluar nuestras habilidades en proyectos más cercanos a los desafíos del entorno laboral, nos enseñó la importancia de priorizar, mantener el enfoque y tomar decisiones de manera ágil. Esta experiencia también nos mostró la importancia de aprender a trabajar bajo presión y con el tiempo en contra.