**Documentation du Projet OPTIQ ETAPE 3**

Le résultat montre que tout est en ordre pour le nettoyage et la réinitialisation de la base de données. Voici comment nous allons procéder :

**Mise à jour de la documentation**

**Etape 1 pour l’extraction :**

**Gestion des connextions python Code/test\_layer4.py:**import os

from vsdx import VisioFile

def analyze\_activity\_connections(shapes, shape\_by\_id):

"""

Analyse les connexions pour chaque activité.

"""

activity\_data = {}

special\_shapes = {}

for shape in shapes:

layer = shape.xml.find(".//{\*}Cell[@N='LayerMember']")

layer\_value = layer.get("V") if layer is not None else "None"

# Identifier les activités principales

if layer\_value == "1":

activity\_data[shape.ID] = {

"name": shape.text.strip(),

"declenchante\_in": [],

"nourrissante\_in": [],

"declenchante\_out": [],

"nourrissante\_out": [],

}

# Identifier les formes spéciales (Retour, Résultat)

elif layer\_value in ["8", "6"]: # Ajouter les layers correspondants

special\_shapes[shape.ID] = {

"name": shape.text.strip(),

"declenchante\_in": [],

"nourrissante\_in": [],

"declenchante\_out": [],

"nourrissante\_out": [],

}

# Identifier les connexions pour les données

elif layer\_value in ["9", "10"]: # Données nourrissantes et déclenchantes

connections = analyze\_connections(shape, shape\_by\_id)

from\_id = connections["from\_id"]

to\_id = connections["to\_id"]

# Traiter les données sortantes

if from\_id in activity\_data:

target\_name = (

special\_shapes[to\_id]["name"] if to\_id in special\_shapes else

activity\_data[to\_id]["name"] if to\_id in activity\_data else "inconnu"

)

if layer\_value == "10":

activity\_data[from\_id]["declenchante\_out"].append(f"{shape.text.strip()} (vers {target\_name})")

elif layer\_value == "9":

activity\_data[from\_id]["nourrissante\_out"].append(f"{shape.text.strip()} (vers {target\_name})")

# Traiter les données entrantes

if to\_id in activity\_data:

source\_name = (

special\_shapes[from\_id]["name"] if from\_id in special\_shapes else

activity\_data[from\_id]["name"] if from\_id in activity\_data else "inconnu"

)

if layer\_value == "10":

activity\_data[to\_id]["declenchante\_in"].append(f"{shape.text.strip()} (depuis {source\_name})")

elif layer\_value == "9":

activity\_data[to\_id]["nourrissante\_in"].append(f"{shape.text.strip()} (depuis {source\_name})")

# Gérer les formes spéciales

if from\_id in special\_shapes:

target\_name = (

activity\_data[to\_id]["name"] if to\_id in activity\_data else "inconnu"

)

if layer\_value == "10":

special\_shapes[from\_id]["declenchante\_out"].append(f"{shape.text.strip()} (vers {target\_name})")

elif layer\_value == "9":

special\_shapes[from\_id]["nourrissante\_out"].append(f"{shape.text.strip()} (vers {target\_name})")

if to\_id in special\_shapes:

source\_name = (

activity\_data[from\_id]["name"] if from\_id in activity\_data else "inconnu"

)

if layer\_value == "10":

special\_shapes[to\_id]["declenchante\_in"].append(f"{shape.text.strip()} (depuis {source\_name})")

elif layer\_value == "9":

special\_shapes[to\_id]["nourrissante\_in"].append(f"{shape.text.strip()} (depuis {source\_name})")

return activity\_data, special\_shapes

def analyze\_connections(shape, shape\_by\_id):

"""

Analyse les connexions d'une forme.

"""

connections = {"from\_id": None, "to\_id": None}

for cell in shape.xml.findall(".//{\*}Cell"):

n\_val = cell.get("N")

f\_val = cell.get("F")

if n\_val == "BeginX":

connections["from\_id"] = f\_val.split("Sheet.")[1].split("!")[0] if "Sheet." in f\_val else None

elif n\_val == "EndX":

connections["to\_id"] = f\_val.split("Sheet.")[1].split("!")[0] if "Sheet." in f\_val else None

return connections

def test\_activities\_with\_special\_shapes():

"""

Test pour analyser les connexions entre activités et formes spéciales.

"""

current\_dir = os.path.dirname(\_\_file\_\_)

vsdx\_path = os.path.join(current\_dir, "example.vsdx")

if not os.path.exists(vsdx\_path):

print(f"ERROR: Fichier example.vsdx introuvable dans '{vsdx\_path}'")

return

print(f"DEBUG: Ouverture du fichier Visio '{vsdx\_path}'")

doc = VisioFile(vsdx\_path)

pages = doc.pages

print(f"DEBUG: Nombre de pages dans ce doc = {len(pages)}")

if not pages:

print("No pages in doc.")

return

page = pages[0]

print(f"DEBUG: On analyse la page 0 nommée '{page.name}'")

shapes = page.all\_shapes

print(f"DEBUG: Nombre de shapes = {len(shapes)}")

# Dictionnaire des shapes par ID

shape\_by\_id = {str(shape.ID): shape for shape in shapes}

# Analyse des connexions et données

activity\_data, special\_shapes = analyze\_activity\_connections(shapes, shape\_by\_id)

# Affichage des résultats

print("\n=== Activités ===")

for activity\_id, data in activity\_data.items():

print(f"Activité : {data['name']} (ID={activity\_id})")

print(f" Données déclenchantes entrantes : {', '.join(data['declenchante\_in']) if data['declenchante\_in'] else 'None'}")

print(f" Données nourrissantes entrantes : {', '.join(data['nourrissante\_in']) if data['nourrissante\_in'] else 'None'}")

print(f" Données déclenchantes sortantes : {', '.join(data['declenchante\_out']) if data['declenchante\_out'] else 'None'}")

print(f" Données nourrissantes sortantes : {', '.join(data['nourrissante\_out']) if data['nourrissante\_out'] else 'None'}")

print("\n=== Formes spéciales (Résultat, Retour) ===")

for shape\_id, data in special\_shapes.items():

print(f"Forme spéciale : {data['name']} (ID={shape\_id})")

print(f" Données déclenchantes entrantes : {', '.join(data['declenchante\_in']) if data['declenchante\_in'] else 'None'}")

print(f" Données nourrissantes entrantes : {', '.join(data['nourrissante\_in']) if data['nourrissante\_in'] else 'None'}")

print(f" Données déclenchantes sortantes : {', '.join(data['declenchante\_out']) if data['declenchante\_out'] else 'None'}")

print(f" Données nourrissantes sortantes : {', '.join(data['nourrissante\_out']) if data['nourrissante\_out'] else 'None'}")

# Fermeture du fichier (pour éviter le PermissionError).

try:

doc.close\_vsdx()

except PermissionError as e:

print(f"PermissionError ignored: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

test\_activities\_with\_special\_shapes()

print("DEBUG: Fin du script test\_activities\_with\_special\_shapes.py")

**gestion des layers** (Venv) PS C:\Users\Hubert.AFDEC\A.F.D.E.C\Projet OPTIQ - DevOPTIQ> Python Code/test\_debug.py

import os

import shutil

from vsdx import VisioFile

def categorize\_layer(layer\_value):

"""Retourne la catégorie d'une forme en fonction de son numéro de layer."""

layer\_categories = {

"1": "Activité",

"10": "Donnée déclenchante",

"9": "Donnée nourrissante",

"6": "Résultat",

"8": "Retour",

}

return layer\_categories.get(layer\_value, "Exclu")

def test\_layer\_extraction\_with\_categories(vsdx\_path):

"""Test pour extraire et catégoriser les formes en fonction de leur layer."""

if not os.path.exists(vsdx\_path):

print(f"Erreur : Le fichier '{vsdx\_path}' est introuvable.")

return

print(f"DEBUG: Ouverture du fichier Visio '{vsdx\_path}'")

try:

with VisioFile(vsdx\_path) as visio:

for page in visio.pages:

print(f"Analyse de la page : {page.name}")

for shape in page.all\_shapes:

layer = shape.xml.find(".//{\*}Cell[@N='LayerMember']")

layer\_value = layer.get("V") if layer is not None else "None"

text = shape.text.strip() if shape.text else "None"

# Catégoriser les layers

category = categorize\_layer(layer\_value)

if category != "Exclu":

print(f"Shape ID={shape.ID}, Text='{text}', Layer='{layer\_value}' ({category})")

else:

print(f"Shape ID={shape.ID}, Text='{text}', Layer='{layer\_value}' (Exclu)")

except PermissionError as e:

print(f"PermissionError ignorée : {e}")

finally:

# Nettoyer explicitement les fichiers temporaires

if visio.directory and os.path.exists(visio.directory):

shutil.rmtree(visio.directory, ignore\_errors=True)

print("Fichiers temporaires nettoyés.")

# Tester le code

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

project\_root = os.path.dirname(current\_dir)

vsdx\_file = os.path.join(project\_root, 'Code', 'example.vsdx')

test\_layer\_extraction\_with\_categories(vsdx\_file)

1. **Fichiers testés et validés :**
   * test\_import.py
   * test\_db\_interaction.py
   * Sup\_doublons.py
   * reset\_database.py  
       
       
     Ces test ont été réalisés avec une structure dans models qui était différentes :
2. from Code.extensions import db
3. class Activity(db.Model):
4. """Table pour les activités."""
5. \_\_tablename\_\_ = 'activities'
6. id = db.Column(db.Integer, primary\_key=True)
7. name = db.Column(db.String(255), unique=True, nullable=False)
8. description = db.Column(db.Text, nullable=True)
9. class Data(db.Model):
10. """Table pour les données (déclenchantes ou nourrissantes)."""
11. \_\_tablename\_\_ = 'data'
12. id = db.Column(db.Integer, primary\_key=True)
13. name = db.Column(db.String(255), unique=True, nullable=False)
14. type = db.Column(db.String(50), nullable=False)  # Ex.: "nourrissante", "déclenchante"
15. description = db.Column(db.Text, nullable=True)
16. class Connection(db.Model):
17. """Table pour les connexions entre activités et données."""
18. \_\_tablename\_\_ = 'connections'
19. id = db.Column(db.Integer, primary\_key=True)
20. source\_id = db.Column(db.Integer, db.ForeignKey('data.id'), nullable=False)
21. target\_id = db.Column(db.Integer, db.ForeignKey('activities.id'), nullable=False)
22. type = db.Column(db.String(50), nullable=False)  # Ex.: "input", "output"
23. description = db.Column(db.Text, nullable=True)
24. **Leçons apprises :**
    * **Gestion des chemins relatifs** :
      + Ajout explicite des chemins nécessaires dans sys.path avant les importations pour éviter les erreurs liées au module Code.
    * **Accès à la base de données** :
      + Utilisation correcte de l'objet SQLAlchemy avec le contexte de l'application Flask pour éviter les erreurs liées à Engine ou app\_context.
      + Vérification systématique des erreurs d'accès avec des messages explicites.
    * **Nettoyage et réinitialisation** :
      + Suppression des tables inutiles avant de vider les tables essentielles pour éviter les conflits lors de réimportations.

Je vais mettre à jour la documentation et enregistrer ces points.

**Prochaine étape**

prepa\_extract.py

fichier qui remets la base à zéro et ajoute des données

python Code/prepa\_extract.py

table models à  :  
from flask\_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

class Activities(db.Model):

    \_\_tablename\_\_ = 'activities'

    id = db.Column(db.Integer, primary\_key=True)

    name = db.Column(db.String(200), nullable=False)

    description = db.Column(db.Text, nullable=True)

class Connections(db.Model):

    \_\_tablename\_\_ = 'connections'

    id = db.Column(db.Integer, primary\_key=True)

    source\_id = db.Column(db.Integer, nullable=False)

    target\_id = db.Column(db.Integer, nullable=False)

    type = db.Column(db.String(50), nullable=False)

    description = db.Column(db.Text, nullable=True)

class Data(db.Model):

    \_\_tablename\_\_ = 'data'

    id = db.Column(db.Integer, primary\_key=True)

    name = db.Column(db.String(255), nullable=False)

    type = db.Column(db.String(50), nullable=False)

    description = db.Column(db.Text, nullable=True)