**Documentation du projet « OPTIQ – DEVOPTIQ »**

**1. Introduction**

Le projet **OPTIQ – DEVOPTIQ** est une application de traitement et d’extraction de données à partir de fichiers Visio (au format VSDX). L’objectif principal est d’analyser des cartes Visio pour en extraire des informations sur les activités, les données échangées et les connexions entre ces activités. L’application utilise Flask comme framework web, SQLAlchemy pour la gestion de la base de données et la bibliothèque **vsdx** pour la lecture des fichiers Visio.

**2. Fonctionnalités principales**

* **Extraction des informations Visio**  
  L’application parcourt les pages et les formes d’un fichier Visio et extrait les informations suivantes :
  + Les formes d’activité (« Activity » et « Result ») qui sont enregistrées dans la table **Activities**.
  + Les formes de retour (« Return ») qui sont enregistrées dans la table **Data**.
  + Les connecteurs (« T link » pour les données déclenchantes et « N link » pour les données nourrissantes) qui créent des enregistrements dans la table **Data** et relient les activités via la table **Connections**.
  + Un mécanisme spécifique gère les cas particuliers (retours et activités de résultat) afin de refléter correctement le flux de données entre les activités.
* **Gestion des connexions**  
  Les liens entre activités (ou entre activités et retours) sont fusionnés pour fournir un résumé clair des flux :

« Donnée "X" (déclenchante/nourrissante) produite par "Nom de l’activité source" est utilisée par "Nom de l’activité destination". »

* **Base de données**  
  La structure de la base repose sur trois tables principales :
  + **Activities**  
    Enregistre les activités extraites des formes Visio de type « Activity » ou « Result ».  
    Colonnes principales : id, name, description, is\_result (booléen indiquant si l’activité est de résultat).
  + **Data**  
    Enregistre les données extraites des connecteurs et les formes de type « Return » (les retours).  
    Colonnes principales : id, name, type (déclenchante, nourrissante, Retour, etc.), description, layer.
  + **Connections**  
    Enregistre les relations entre activités et données.  
    Colonnes principales : id, source\_id, target\_id, type, description.

**3. Architecture et structure du projet**

Le projet est organisé autour de plusieurs répertoires :

* **Code/**  
  Contient l’ensemble du code source de l’application.
  + **models/**
    - models.py : Définit les modèles SQLAlchemy pour les tables *Activities*, *Data* et *Connections*.
    - \_\_init\_\_.py : (Initialisation du module models).
  + **extensions.py**
    - Initialise et configure SQLAlchemy.
  + **app.py**
    - Point d’entrée de l’application Flask.
  + **routes/**
    - Contient les fichiers définissant les routes de l’application (par exemple, pour afficher ou interagir avec les activités).
  + **scripts/**
    - extract\_visio.py : Script principal pour l’extraction et le traitement des fichiers Visio.
    - D’autres scripts utilitaires et de test (ex. : check\_db\_structure.py, prepa\_extract.py, etc.) qui ont été utilisés en phase de développement et peuvent être archivés si non indispensables en production.
  + **static/** et **Templates/**
    - Contiennent respectivement les ressources statiques (CSS, JS, images) et les templates HTML utilisés par l’interface utilisateur.
* **migrations/**  
  Gère les migrations de base de données avec Alembic (fichiers alembic.ini, env.py, et le répertoire versions/).
* **Venv/**  
  L’environnement virtuel Python contenant toutes les dépendances du projet.
* **Bases de données/**  
  Ce répertoire (ou un sous-répertoire dans Code/instance) contient le fichier de la base SQLite (optiq.db).
* **Archives/**  
  Répertoire (optionnel) dans lequel sont déplacés les fichiers et scripts de tests ou les fichiers obsolètes que l’on ne souhaite plus garder dans le dépôt principal.
* **Autres fichiers à la racine**
  + .gitignore : Liste des fichiers/répertoires à ignorer par Git (ex. l’environnement virtuel, la base de données, certains répertoires de tests ou temporaires, etc.).
  + requirements.txt : Liste des dépendances Python du projet.
  + start\_environment.bat : Script pour démarrer l’environnement (configuration de variables d’environnement, lancement de l’application, etc.).
  + D’autres fichiers utilitaires (documents, sauvegardes, etc.) pouvant être archivés.

**4. Description détaillée de certains fichiers et leur utilité**

**Fichiers de scripts d’extraction et tests**

* **extract\_visio.py**  
  Ce script lit un fichier Visio (VSDX) et effectue l’extraction des formes selon leur calque (activités, résultats, retours, connecteurs).  
  Il utilise des mappings globaux pour associer les identifiants Visio aux identifiants en base (pour les tables *Activities* et *Data*).  
  Le script crée également les connexions entre ces éléments et génère un résumé des connexions.
* **check\_db.py / inspect\_db.py**  
  Scripts utilitaires permettant de vérifier la structure et l’état de la base de données (tables existantes, colonnes, etc.).  
  Ils sont utiles lors du développement pour s’assurer que les migrations se sont bien appliquées.
* **recreate\_tables.sql**  
  Fichier SQL permettant de recréer manuellement les tables principales (*activities*, *connections*, *data*).  
  Ce script a été utilisé pendant le développement pour tester la création des tables, mais désormais, le projet utilise SQLAlchemy et les migrations Alembic. Vous pouvez archiver ce fichier s’il n’est plus nécessaire en production.

**Fichiers de configuration et initialisation**

* **app.py**  
  Initialise l’application Flask, configure les extensions (SQLAlchemy, etc.) et définit les routes principales.
* **extensions.py**  
  Contient l’instance de SQLAlchemy et éventuellement d’autres extensions Flask utilisées par le projet.
* **migrations/**  
  Ce répertoire contient la configuration et les scripts de migration (générés par Flask-Migrate/Alembic) qui permettent de faire évoluer la structure de la base de données au fil du développement.

**Fichiers et dossiers d’archives**

* **Archives/**  
  Ce dossier regroupe les scripts et documents anciens ou obsolètes (ex. : tests intermédiaires, anciennes versions de scripts d’extraction, documents de tests).  
  Vous devez conserver ce dossier séparément pour référence, mais il n’interfère pas avec le fonctionnement de l’application.

**Fichiers divers**

* **exported\_data.csv**  
  Ce fichier est généré lors de l’export ou de la vérification de l’extraction des données.  
  Il peut être utilisé pour des vérifications ou des analyses manuelles, mais il n’est pas directement utilisé par l’application en production.
* **structure\_projet.txt**  
  Fichier descriptif (généré ou maintenu manuellement) qui présente la structure du projet.  
  Il sert de guide pour comprendre l’organisation des fichiers et répertoires.

**5. Instructions d’installation et d’exécution**

1. **Installation de l’environnement**
   * Clonez le dépôt depuis GitHub.
   * Créez un environnement virtuel :

bash

Copier

python -m venv Venv

* + Activez l’environnement virtuel :
    - Sur Windows : Venv\Scripts\activate
    - Sur Linux/Mac : source Venv/bin/activate
  + Installez les dépendances :

bash

Copier

pip install -r requirements.txt

1. **Configuration**
   * Assurez-vous que le fichier .gitignore contient les répertoires à ignorer (ex. : Venv/, Code/instance/, etc.).
   * Le fichier start\_environment.bat (ou un équivalent pour votre système) peut être utilisé pour configurer les variables d’environnement et démarrer l’application.
2. **Migrations et Base de données**
   * Pour initialiser la base de données, exécutez :

bash

Copier

flask db upgrade

* + Vous pouvez vérifier la structure de la base à l’aide du script inspect\_db.py.

1. **Exécution de l’application**
   * Démarrez l’application avec Flask (par exemple, via flask run) ou exécutez directement le script principal selon la configuration du projet.
2. **Utilisation de l’extraction Visio**
   * Placez votre fichier Visio (ex. : example.vsdx) dans le répertoire approprié.
   * Exécutez le script extract\_visio.py pour extraire les données et générer le résumé des connexions.

**6. Conclusion**

Ce document résume l’état actuel du projet **OPTIQ – DEVOPTIQ** ainsi que la fonction de chacun des composants principaux. La structure du projet permet de maintenir une séparation claire entre le code source de l’application, les scripts utilitaires et les éléments de tests ou archives.

En cas de besoin, les fichiers de tests et de vérification (comme inspect\_db.py, check\_db.py, ou recreate\_tables.sql) peuvent être archivés ou ajoutés au .gitignore pour éviter qu’ils ne soient inclus dans la version de production.