**Introduction**

L’application OP ITQ (nom fictif pour cet exemple) est une application web développée en Python avec le framework Flask. Elle permet de :

* Extraire et traiter des données à partir d’une cartographie Visio (via le script extract\_visio.py),
* Gérer des activités, tâches et outils dans une base de données SQLite via SQLAlchemy,
* Offrir une interface ergonomique pour consulter, ajouter, supprimer et réordonner les éléments grâce à une interface web moderne (avec Font Awesome et SortableJS).

**Architecture du projet**

L’arborescence du projet est la suivante :

Projet OPTIQ - DevOPTIQ/

├─ Archives/

├─ Bases de données/

├─ Code/

│ ├─ app.py # Point d’entrée de l’application Flask

│ ├─ base\_sql.py # Configuration de la base SQL et diagnostics

│ ├─ extensions.py # Initialisation de l’instance db pour SQLAlchemy

│ ├─ models/

│ │ └─ models.py # Définition des modèles (Activities, Connections, Data, Task, Tool)

│ ├─ migrations/ # Scripts de migration Alembic

│ ├─ routes/

│ │ ├─ activities.py # Routes pour gérer activités, tâches et mise à jour de la cartographie

│ │ ├─ tools.py # Routes pour gérer l’ajout et la suppression d’outils

│ │ └─ templates/

│ │ └─ activities\_list.html # Template HTML pour l’affichage des activités, tâches et outils

│ ├─ scripts/

│ │ └─ extract\_visio.py # Script pour extraire et traiter les données de la cartographie Visio

│ └─ instance/ # Contient la base de données (optiq.db)

├─ requirements.txt # Liste des dépendances du projet

└─ ...

**Description des principaux fichiers**

**a) Code/app.py**

* C’est le point d’entrée de l’application Flask.
* Il configure l’application (chemin de la base de données, track modifications, etc.), initialise SQLAlchemy et Flask-Migrate, et enregistre les blueprints pour les routes d’activités et d’outils.

**b) Code/extensions.py**

* Contient l’instance de SQLAlchemy (db = SQLAlchemy()).
* Est importé par tous les modules nécessitant l’accès à la base de données.

**c) Code/models/models.py**

* Définit les modèles de données :
  + **Activities** : Pour stocker les activités.
  + **Connections** : Pour stocker les relations entre activités.
  + **Data** : Pour stocker les informations extraites (ex. depuis Visio).
  + **Task** : Pour gérer les tâches associées aux activités.
  + **Tool** : Pour gérer les outils associés aux tâches.

**d) Code/routes/activities.py**

* Contient les endpoints pour :
  + Créer et afficher les activités.
  + Ajouter et supprimer des tâches.
  + Réordonner les tâches (avec l’ordre sauvegardé dans le champ order).
  + Mettre à jour la cartographie en appelant le script extract\_visio.py.
* Lors de l’affichage, il assemble les données (activités, connexions, tâches et outils) pour le template activities\_list.html.

**e) Code/routes/tools.py**

* Contient les endpoints pour :
  + Ajouter des outils à une tâche (en choisissant parmi les outils existants ou en ajoutant de nouveaux).
  + Supprimer un outil d’une tâche.
  + Fournir la liste des outils existants.

**f) Code/routes/templates/activities\_list.html**

* Le template HTML qui affiche l’interface utilisateur.
* Permet de visualiser les activités, leurs tâches, et les outils associés.
* Offre des boutons (icônes) pour ajouter, supprimer et réordonner les tâches, ainsi que pour ajouter et supprimer les outils.
* Intègre SortableJS pour le drag & drop des tâches et Font Awesome pour les icônes.

**g) Code/scripts/extract\_visio.py**

* Un script qui extrait les données d’un fichier Visio.
* Utilise la bibliothèque vsdx pour lire le fichier Visio.
* Insère ou met à jour les données dans la base (activités, connexions, données).
* Dispose de fonctions comme reset\_database(), process\_visio\_file(vsdx\_path) et print\_summary() pour orchestrer l’extraction.

**Utilisation et déploiement**

1. **Installation des dépendances**  
   Assurez-vous que toutes les dépendances listées dans requirements.txt sont installées dans votre environnement virtuel.
2. **Exécution de l’application**  
   Depuis la racine du projet, lancez l’application avec :

python Code/app.py

L’application s’exécutera en mode debug et sera accessible sur http://127.0.0.1:5000.

1. **Mise à jour de la cartographie**  
   Depuis l’interface, cliquez sur le bouton vert « Mettre à jour la cartographie » pour lancer l’extraction du fichier Visio et actualiser la base de données.
2. **Gestion des activités, tâches et outils**  
   Utilisez l’interface pour ajouter, supprimer et réordonner les tâches, ainsi que pour associer des outils aux tâches.

**Commandes Git**

* Pour fusionner le commit « Opérationnel bouton et glisser deplacer » dans main, suivez ces commandes :

git checkout main

git merge <nom-de-la-branche>

git push origin main

(ou utilisez git cherry-pick <commit-id> si vous souhaitez prendre uniquement ce commit.)

**Perspectives d’évolution**

* Implémenter la sauvegarde permanente de l’ordre des tâches dans la base (l’endpoint /tasks/reorder est en place, à intégrer dans l’interface).
* Ajouter des fonctionnalités d’édition pour les activités, tâches et outils.
* Améliorer l’interface avec des animations ou une meilleure gestion responsive.

Cette documentation devrait servir de référence complète pour notre application. N’hésitez pas à me faire savoir si vous souhaitez ajouter d’autres sections ou si vous avez des questions pour poursuivre le développement dans un futur chat.