**Documentation du Projet OPTIQ ETAPE 2**

**Structure du Projet**

**Arborescence des Fichiers**

Projet OPTIQ - DevOPTIQ/

├── Archives/

├── Bases de données/

├── Code/

│ ├── instance/

│ │ └── optiq.db

│ ├── models/

│ │ ├── \_\_init\_\_.py

│ │ └── models.py

│ ├── routes/

│ │ ├── \_\_init\_\_.py

│ │ ├── activities.py

│ │ └── ui\_routes.py

│ ├── scripts/

│ │ ├── \_\_init\_\_.py

│ │ ├── extract\_visio.py

│ │ └── insert\_activities.py

│ ├── static/ui/

│ │ └── Templates/

│ ├── add\_activities.py

│ │ app.py

│ │ create\_tables.py

│ │ extensions.py

│ │ initialize\_db.py

│ │ example.vsdx

├── Temp/instance/

├── Venv/

│ └── Include/

│ └── Lib/

│ └── Scripts/

├── .gitignore

├── requirements.txt

├── start\_environment.bat

└── structure\_projet.txt

**Détails des Fichiers et Leur Usage**

**1. Base de données**

* `` : La base de données SQLite contenant les activités et relations importées depuis le fichier Visio.

**2. Code/instance**

* **Usage** : Contient les fichiers essentiels au fonctionnement local du projet, comme la base de données optiq.db.

**3. Code/models**

* `` :
  + Définit les modèles de données SQLAlchemy :
    - Activity : Représente une activité.
    - Relation : Représente les relations entre les activités.

**4. Code/routes**

* `` : Gère les routes pour les activités dans l'application Flask.
* `` : Gère les routes de l'interface utilisateur.

**5. Code/scripts**

* `` :
  + Fonction principale : Importer des activités et relations depuis un fichier Visio .vsdx.
  + Crée ou met à jour les activités dans la base de données.
* `` :
  + Insère manuellement des activités dans la base pour des tests ou des usages spécifiques.

**6. Code/extensions.py**

* **Usage** : Initialise l'extension SQLAlchemy pour la gestion de la base de données.

**7. Code/initialize\_db.py**

* **Usage** :
  + Initialise la base de données SQLite et ses tables.
  + Insère des activités de test pour valider la configuration.

**8. Autres fichiers principaux**

* `` : Point d'entrée principal de l'application Flask.
* `` : Fichier Visio d'exemple pour les tests d'importation.
* `` : Liste des dépendances Python du projet.
* `` : Script pour initialiser l'environnement de travail.

**Fichier Batch mis à jour :**

@echo off

REM Script pour configurer et lancer l'environnement de développement Flask.

REM 1. Définir le répertoire racine de votre projet.

set PROJECT\_DIR=C:\Users\Hubert.AFDEC\A.F.D.E.C\Projet OPTIQ - DevOPTIQ

REM 2. Aller dans le répertoire du projet.

cd "%PROJECT\_DIR%"

REM 3. Vérification des outils nécessaires.

echo Vérification de la version de Git...

git --version

if errorlevel 1 (

echo Erreur : Git n'est pas trouvé. Installez Git ou ajoutez-le au PATH.

pause

exit /b 1

)

echo Vérification de la version de Python...

python --version

if errorlevel 1 (

echo Erreur : Python n'est pas trouvé. Installez Python ou ajoutez-le au PATH.

pause

exit /b 1

)

REM 4. Activer l'environnement virtuel Python.

echo Activation de l'environnement virtuel Python...

call "%PROJECT\_DIR%\Venv\Scripts\activate.bat"

if errorlevel 1 (

echo Erreur : Impossible d'activer l'environnement virtuel. Vérifiez le dossier Venv.

pause

exit /b 1

)

REM 5. Installer les dépendances.

echo Installation/mise à jour des dépendances Python...

pip install -r "%PROJECT\_DIR%\requirements.txt"

if errorlevel 1 (

echo Erreur : Échec de l'installation des dépendances. Vérifiez requirements.txt.

pause

exit /b 1

)

REM 6. Vérification des chemins de fichiers critiques.

if not exist "%PROJECT\_DIR%\Code\app.py" (

echo Erreur : Le fichier app.py est introuvable dans le répertoire Code.

pause

exit /b 1

)

REM 7. Configurer Flask pour le lancement.

echo Configuration de l'application Flask...

set FLASK\_APP=app.py

set FLASK\_ENV=development

REM 8. Naviguer dans le répertoire contenant le code.

cd "%PROJECT\_DIR%\Code"

REM 9. Lancer l'application Flask.

echo Lancement de l'application Flask...

flask run

if errorlevel 1 (

echo Erreur : Échec du lancement de l'application Flask.

pause

exit /b 1

)

REM 10. Pause pour maintenir la fenêtre ouverte en cas d'erreur.

pause

**Leçons Apprises et Améliorations à Apporter**

**1. Gestion des chemins et environnement**

* Centralisation et validation stricte des chemins relatifs et absolus.
* Utilisation de os.path.abspath dans tous les fichiers pour éviter les erreurs.

**2. Validation incrémentale**

* Travailler par étapes simples.
* Tester chaque modification avant d'aller plus loin.

**3. Documentation rigoureuse**

* Maintenir une documentation exhaustive des changements, erreurs et solutions.
* Introduire une section "Journal des erreurs".

**4. Focus sur les objectifs prioritaires**

* Planifier les objectifs à court terme (par exemple, gestion des activités dans la base de données) avant de passer à des améliorations secondaires.

**5. Automatisation des tests**

* Ajouter des scripts pour valider automatiquement des points critiques (connexion à la base de données, insertion d'activités, etc.).
* Exemple : un script test\_db.py pour valider la configuration avant l'exécution du projet.

**Prochaines Étapes**

* Finaliser l'importation des relations entre activités dans extract\_visio.py.
* Nettoyer les doublons dans la base de données.
* Mettre en place une interface utilisateur pour visualiser les activités et relations importées.
* Documenter les étapes réalisées et leurs résultats.

**Annexe : détail des fichiers :**

**Code/models/Models.py**

from Code.extensions import db

class Activity(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)

    incoming\_relations = db.relationship('Relation', foreign\_keys='Relation.target\_id', backref='target\_activity', lazy=True)

    outgoing\_relations = db.relationship('Relation', foreign\_keys='Relation.source\_id', backref='source\_activity', lazy=True)

class Relation(db.Model):

    \_\_tablename\_\_ = 'relations'

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

    source\_id = db.Column(db.Integer, db.ForeignKey('activities.id'), nullable=False)

    target\_id = db.Column(db.Integer, db.ForeignKey('activities.id'), nullable=False)

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

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

**Code/scripts/extract\_visio.py**import os

import sys

from vsdx import VisioFile

from flask import Flask

# Ajouter dynamiquement le répertoire racine du projet dans sys.path

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

project\_root = os.path.dirname(os.path.dirname(current\_dir))  # Racine du projet

if project\_root not in sys.path:

    sys.path.insert(0, project\_root)

# Import après avoir ajouté le chemin

try:

    from Code.extensions import db

    from Code.models.models import Activity, Relation

except ImportError as e:

    print(f"Erreur d'import : {e}")

    print("Chemins disponibles dans sys.path :")

    for path in sys.path:

        print(path)

    sys.exit(1)

def create\_app():

    """Initialise l'application Flask et configure la base de données."""

    app = Flask(\_\_name\_\_)

    app.config['SQLALCHEMY\_DATABASE\_URI'] = f"sqlite:///{os.path.abspath(os.path.join(project\_root, 'Code', 'instance', 'optiq.db')).replace('\\', '/')}"

    app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

    db.init\_app(app)

    return app

def import\_from\_visio(vsdx\_path):

    """Importe les données d'un fichier Visio dans la base SQLite."""

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

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

        return

    with VisioFile(vsdx\_path) as visio:

        for page in visio.pages:

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

            for shape in page.child\_shapes:

                shape\_text = shape.text.strip() if shape.text else None

                # Ignorer les formes sans texte pertinent

                if not shape\_text:

                    continue

                # Ajouter ou mettre à jour une activité

                activity = \_create\_or\_update\_activity(shape\_text)

                # Analyser les connexions

                for connection in shape.connects:

                    try:

                        # Vérifier si la connexion a un attribut 'to\_shape'

                        if hasattr(connection, "to\_shape") and connection.to\_shape:

                            target\_text = connection.to\_shape.text.strip() if connection.to\_shape.text else None

                        else:

                            target\_text = None

                        source\_text = shape\_text

                        if target\_text:

                            \_create\_or\_update\_relation(source\_text, target\_text)

                    except Exception as e:

                        print(f"Erreur lors de l'analyse d'une connexion : {e}")

                        continue

def \_create\_or\_update\_activity(activity\_text):

    """Ajoute ou met à jour une activité dans la base."""

    existing = Activity.query.filter\_by(name=activity\_text).first()

    if not existing:

        new\_activity = Activity(name=activity\_text)

        db.session.add(new\_activity)

        db.session.commit()

        print(f"Nouvelle activité ajoutée : {activity\_text}")

        return new\_activity

    else:

        print(f"Activité existante trouvée : {activity\_text}")

        return existing

def \_create\_or\_update\_relation(source\_text, target\_text):

    """Ajoute ou met à jour une relation entre deux activités."""

    source\_activity = Activity.query.filter\_by(name=source\_text).first()

    target\_activity = Activity.query.filter\_by(name=target\_text).first()

    if not source\_activity or not target\_activity:

        print(f"Impossible de créer la relation : Source '{source\_text}' ou Target '{target\_text}' non trouvée.")

        return

    # Vérifier si la relation existe déjà

    existing\_relation = Relation.query.filter\_by(source\_id=source\_activity.id, target\_id=target\_activity.id).first()

    if not existing\_relation:

        new\_relation = Relation(

            source\_id=source\_activity.id,

            target\_id=target\_activity.id,

            type="unknown",

            description=f"Relation entre {source\_text} et {target\_text}"

        )

        db.session.add(new\_relation)

        db.session.commit()

        print(f"Nouvelle relation ajoutée : {source\_text} -> {target\_text}")

    else:

        print(f"Relation existante entre {source\_text} et {target\_text}.")

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

    # Initialiser l'application Flask

    app = create\_app()

    # Créer les tables si elles n'existent pas

    with app.app\_context():

        db.create\_all()

        print("Tables créées ou mises à jour avec succès.")

        # Chemin du fichier Visio

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

        if os.path.exists(vsdx\_file):

            import\_from\_visio(vsdx\_file)

        else:

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

**Code/app.py**

import os

import sys

# Ajouter dynamiquement le répertoire parent et 'Code' dans sys.path

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

parent\_dir = os.path.dirname(current\_dir)

if current\_dir not in sys.path:

    sys.path.insert(0, current\_dir)

if parent\_dir not in sys.path:

    sys.path.insert(0, parent\_dir)

from Code.extensions import db

from Code.models.models import Activity

from Code.routes.ui\_routes import ui\_bp

from Code.routes.activities import activities\_bp

from flask import Flask, render\_template

from sqlalchemy import text

def create\_app():

    app = Flask(\_\_name\_\_)

    # Configuration de la base de données

    instance\_path = os.path.join(os.path.dirname(\_\_file\_\_), 'instance')

    db\_path = os.path.join(instance\_path, 'optiq.db')

    app.config['SQLALCHEMY\_DATABASE\_URI'] = f"sqlite:///{db\_path}"

    app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

    # Création du dossier 'instance' si nécessaire

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

        os.makedirs(instance\_path)

    # Initialisation des extensions

    db.init\_app(app)

    # Enregistrement des blueprints

    app.register\_blueprint(activities\_bp)

    app.register\_blueprint(ui\_bp)

    return app

# Déclarer explicitement 'app' pour l'import

app = create\_app()

if \_\_name\_\_ == '\_\_main\_\_':

    with app.app\_context():

        try:

            # Vérification de la connexion à la base de données

            db.session.execute(text('SELECT 1'))

            print("Connexion à la base de données réussie.")

        except Exception as e:

            print(f"Erreur lors de la connexion à la base de données : {e}")

        # Création ou mise à jour des tables

        db.create\_all()

        print("Tables créées ou mises à jour avec succès !")

    app.run(debug=True)

**Code/initialize\_db.py**

import os

import sys

# Ajouter dynamiquement le répertoire racine du projet dans sys.path

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

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

if project\_root not in sys.path:

    sys.path.insert(0, project\_root)

# Afficher les chemins actuels pour vérifier

print("Chemins actuels dans sys.path :")

for path in sys.path:

    print(path)

# Import des modules nécessaires

from Code.extensions import db

from Code.scripts.extract\_visio import create\_app

from Code.models.models import Activity  # Pour tester l'insertion

# Chemin absolu vers la base de données

db\_directory = os.path.join(current\_dir, "instance")

db\_path = os.path.join(db\_directory, "optiq.db")

# Créer le répertoire 'instance' s'il n'existe pas

if not os.path.exists(db\_directory):

    os.makedirs(db\_directory)

    print(f"Répertoire créé : {db\_directory}")

# Vérifier si le fichier de base de données existe déjà

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

    print(f"Base de données non trouvée. Elle sera créée à l'emplacement : {db\_path}")

else:

    print(f"Base de données existante détectée : {db\_path}")

# Initialisation de l'application Flask et configuration

app = create\_app()

app.config['SQLALCHEMY\_DATABASE\_URI'] = f"sqlite:///{os.path.abspath(db\_path).replace('\\', '/')}"

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

print(f"SQLAlchemy URI utilisé : {app.config['SQLALCHEMY\_DATABASE\_URI']}")

# Test de persistance explicite avec SQLAlchemy (ajouté ici)

from sqlalchemy import create\_engine, text

print("Test de persistance avec SQLAlchemy...")

engine = create\_engine(app.config['SQLALCHEMY\_DATABASE\_URI'])

try:

    with engine.connect() as connection:

        # Création de la table de test

        connection.execute(text("CREATE TABLE IF NOT EXISTS persistence\_test (id INTEGER PRIMARY KEY, name TEXT)"))

        # Insertion dans la table de test

        connection.execute(text("INSERT INTO persistence\_test (name) VALUES (:name)"), {"name": "Persist Test"})

        # Lecture des données insérées

        result = connection.execute(text("SELECT id, name FROM persistence\_test"))

        rows = [{"id": row[0], "name": row[1]} for row in result]  # Accès par indices

        print(f"Résultats de la table 'persistence\_test': {rows}")

except Exception as e:

    print(f"Erreur lors du test de persistance : {e}")

finally:

    engine.dispose()

# Initialiser la base de données

try:

    with app.app\_context():

        print("Tentative de création des tables...")

        db.create\_all()

        print("Tables créées avec succès.")

except Exception as e:

    print(f"Erreur critique lors de la création des tables : {e}")

    import traceback

    traceback.print\_exc()

# Vérification du fichier SQLite

db\_physical\_path = os.path.abspath(db\_path)

print(f"Vérification physique du fichier SQLite : {db\_physical\_path}")

if os.path.exists(db\_physical\_path):

    print(f"Fichier de base de données trouvé à : {db\_physical\_path}")

else:

    print(f"Erreur : Le fichier de base de données n'existe pas à : {db\_physical\_path}")

# Test d'insertion pour vérifier la fonctionnalité de la base de données

try:

    with app.app\_context():

        print("Insertion de test dans la table 'activities'...")

        test\_activity = Activity(name="Test Activity", description="Validation de la base de données")

        db.session.add(test\_activity)

        db.session.commit()

        print("Insertion de test réussie.")

except Exception as e:

    print(f"Erreur lors de l'insertion de test : {e}")

    import traceback

    traceback.print\_exc()

# Vérification des données insérées

try:

    with app.app\_context():

        print("Récupération des données insérées dans la table 'activities'...")

        activities = Activity.query.all()

        if activities:

            print(f"Données récupérées : {[{'id': a.id, 'name': a.name, 'description': a.description} for a in activities]}")

        else:

            print("Aucune donnée trouvée dans la table 'activities'.")

except Exception as e:

    print(f"Erreur lors de la récupération des données : {e}")

    import traceback

    traceback.print\_exc()

**Code/extension.py**

from flask\_sqlalchemy import SQLAlchemy

db = SQLAlchemy()

**Code/routes/ui\_routes.py**

from flask import Blueprint, render\_template

from Code.models.models import Activity

ui\_bp = Blueprint('ui', \_\_name\_\_, url\_prefix='/ui')

@ui\_bp.route('/activities', methods=['GET'])

def activities():

    # Récupérer toutes les activités depuis la base de données

    activities = Activity.query.all()

    return render\_template('ui/activities.html', activities=activities)

**Code/routes/activities.py**

import sys

print("sys.path:", sys.path)  # Debug : Vérifier les chemins accessibles

from flask import Blueprint, jsonify, request

from Code.extensions import db

from Code.models.models import Activity

activities\_bp = Blueprint('activities', \_\_name\_\_)

@activities\_bp.route('/activities', methods=['GET'])

def get\_activities():

    activities = Activity.query.all()

    return jsonify([{"id": activity.id, "name": activity.name} for activity in activities])

@activities\_bp.route('/activities', methods=['POST'])

def create\_activity():

    data = request.get\_json()

    new\_activity = Activity(name=data['name'])

    db.session.add(new\_activity)

    db.session.commit()

    return jsonify({"id": new\_activity.id, "name": new\_activity.name}), 201

**Code/Templates/ui/activities.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Liste des activités</title>

    <style>

        .activity-block {

            border: 1px solid #ccc;

            margin: 10px 0;

            padding: 10px;

            border-radius: 5px;

            background-color: #f9f9f9;

        }

        .activity-header {

            cursor: pointer;

            font-weight: bold;

            color: #007BFF;

        }

        .activity-details {

            display: none;

            margin-top: 10px;

        }

    </style>

</head>

<body>

    <h1>Liste des activités</h1>

    <div id="activities-container">

        {% for activity in activities %}

            <div class="activity-block">

                <div class="activity-header" onclick="toggleDetails('{{ activity.id }}')">

                    {{ activity.name }}

                </div>

                <div class="activity-details" id="details-{{ activity.id }}">

                    <p>Description : {{ activity.description or 'Aucune description' }}</p>

                    <h4>Données déclenchantes :</h4>

                    <ul>

                        {% for relation in activity.incoming\_relations %}

                            <li>{{ relation.source\_activity.name }} - {{ relation.description }}</li>

                        {% endfor %}

                    </ul>

                    <h4>Données nourrissantes :</h4>

                    <ul>

                        {% for relation in activity.outgoing\_relations %}

                            <li>{{ relation.target\_activity.name }} - {{ relation.description }}</li>

                        {% endfor %}

                    </ul>

                </div>

            </div>

        {% endfor %}

    </div>

    <script>

        function toggleDetails(activityId) {

            const detailsElement = document.getElementById(`details-${activityId}`);

            if (detailsElement.style.display === "none" || detailsElement.style.display === "") {

                detailsElement.style.display = "block";

            } else {

                detailsElement.style.display = "none";

            }

        }

    </script>

</body>

</html>