Export de l'Architecture, du Code et de la Structure de la Base de Données

1. Architecture du Projet

- 📁 Code
  - __init__.py
  - app.py
  - base_sql.py
  - extensions.py
  - 📁 instance
  - 📁 models
    - __init__.py
    - models.py
  - 📁 routes
    - __init__.py
    - activities.py
    - roles.py
    - roles_view.py
    - skills.py
    - softskills.py
    - tasks.py
    - 📁 templates
      - activities_list.html
      - activity_competencies.html
      - activity_connections.html

```
      activity_header.html
      activity_softskills.html
      activity_tasks.html
      competency_modal.html
      display_list.html
      roles_modal.html
      roles_view.html
      skills_section.html
      translate_softskills_modal.html
    tools.py
    ui_routes.py
  scripts
    __init__.py
    extract_visio.py
Dox projet
export_code.py
migrations
  env.py
  versions
    37f4fdd5526b_migration_initiale_avec_la_nouvelle_.py
static
  js
    competencies.js
    main.js
```

        roles.js

        softskills.js

        tasks.js

        tools.js

        translate_softskills.js

    optiq.css

2. Code Source

export_code.py

```
import os
from docx import Document
import pydotplus as pydot

# --- IMPORTANT : Adapter ces imports si votre projet est organisé différemment ---
# Ici, on suppose que votre instance SQLAlchemy se trouve dans Code/extensions.py
# et que vos modèles sont définis dans Code/models/models.py (utilisant db.Model).
from Code.extensions import db
import Code.models.models  # Ceci permet d'enregistrer vos modèles
# -----------------------------------------------------------------------------

# Création du document Word
doc = Document()
doc.add_heading("Export de l'Architecture, du Code et de la Structure de la Base de
Données", level=1)

# -----------------------------------------------------------------------------
# 1. Architecture du Projet (structure des dossiers et fichiers)
# -----------------------------------------------------------------------------
doc.add_heading("1. Architecture du Projet", level=2)
root_dir = os.getcwd()
excluded_dirs = {"Archives", "Venv", "__pycache__", ".git"}

def add_architecture(path, indent=""):
    try:
        items = sorted(os.listdir(path))
    except Exception as e:
        doc.add_paragraph(f"{indent}Erreur en listant {path}: {e}")
        return
    for item in items:
        # Ignorer les fichiers cachés
        if item.startswith("."):
            continue
        full_path = os.path.join(path, item)
        if os.path.isdir(full_path) and item not in excluded_dirs:
            doc.add_paragraph(f"{indent}📁 {item}")
            add_architecture(full_path, indent + "    ")
        elif os.path.isfile(full_path) and item.endswith((".py", ".js", ".html", ".css")):
            doc.add_paragraph(f"{indent}📄 {item}")

add_architecture(root_dir)

# ---------------------------------------------------------------------------
# 2. Code Source (contenu des fichiers de code utiles)
# ---------------------------------------------------------------------------
doc.add_heading("2. Code Source", level=2)
for folder, _, files in os.walk(root_dir):
    if any(excl in folder for excl in excluded_dirs):
        continue  # Ignorer certains dossiers
    for file in files:
        if file.endswith((".py", ".js", ".html", ".css")):
            file_path = os.path.join(folder, file)
            doc.add_heading(f"{file}", level=3)
            try:
                with open(file_path, "r", encoding="utf-8") as f:
                    code = f.read()
                    doc.add_paragraph(code)
            except Exception as e:
                doc.add_paragraph(f"❌ Erreur de lecture pour {file}: {e}")

# ---------------------------------------------------------------------------
# 3. Structure de la Base de Données (texte)
# ---------------------------------------------------------------------------
doc.add_heading("3. Structure de la Base de Données (Texte)", level=2)
metadata = db.Model.metadata  # Récupère la métadonnée globale de vos modèles

for table in metadata.sorted_tables:
    doc.add_heading(f"Table: {table.name}", level=3)
    for column in table.columns:
        doc.add_paragraph(f"• {column.name} ({column.type})")

# ---------------------------------------------------------------------------
# 4. Diagramme Visuel de la Base de Données (avec relations entre tables)
# ----------------------------------------------------------------------------
doc.add_heading("4. Diagramme Visuel de la Base de Données", level=2)

def create_schema_graph(metadata):
    """
    Crée un diagramme relationnel (ERD) où chaque table est un seul noeud
    listant ses colonnes, et où les foreign keys relient les tables entre elles.
    """
    graph = pydot.Dot(graph_type="digraph", rankdir="LR")

    # Dictionnaire pour référencer les noeuds par nom de table
    table_nodes = {}

    # 1️⃣ Créer un noeud "record" pour chaque table
    for table in metadata.tables.values():
        # Label : { tableName | col1 : type1\ncol2 : type2\n... }
        label = f"{{ {table.name} | "
        cols = [f"{col.name} : {col.type}" for col in table.columns]
        # \l force un saut de ligne "left-justified"
        label += "\\l".join(cols) + "\\l}"

        node = pydot.Node(
            table.name,
            shape="record",
            label=label,
            style="filled",
            fillcolor="lightblue"
        )
        graph.add_node(node)
        table_nodes[table.name] = node

    # 2️⃣ Ajouter des arêtes pour chaque Foreign Key
    for table in metadata.tables.values():
        for fk in table.foreign_keys:
            referencing_node = table_nodes[table.name]
            referenced_node = table_nodes[fk.column.table.name]
            # Indiquer la colonne qui fait la FK
            edge_label = f"{fk.parent.name} → {fk.column.name}"

            edge = pydot.Edge(
                referencing_node,
                referenced_node,
                label=edge_label,
            color="blue",
            arrowsize="0.8"
        )
        graph.add_edge(edge)

    return graph

graph = create_schema_graph(metadata)
diagram_path = os.path.join(root_dir, "database_schema.png")

try:
    graph.write_png(diagram_path)
    doc.add_paragraph(f"Diagramme généré : {diagram_path}")
except Exception as e:
    doc.add_paragraph(f"❌ Erreur lors de la génération du diagramme : {e}")

# ---------------------------------------------------------------------------
# Sauvegarde du document Word
# ---------------------------------------------------------------------------
output_path = os.path.join(root_dir, "export_code.docx")
doc.save(output_path)
print(f"✅ Export terminé ! Fichier généré : {output_path}")
```

app.py

```
# Code/app.py
import os
import sys
from dotenv import load_dotenv

load_dotenv()

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 flask import Flask
from flask_migrate import Migrate
from Code.extensions import db
```

```
def create_app():
    static_folder = os.path.join(parent_dir, 'static')
    app = Flask(__name__, static_folder=static_folder)

    instance_path = os.path.join(os.path.dirname(__file__), 'instance')
    if not os.path.exists(instance_path):
        os.makedirs(instance_path)
    db_path = os.path.join(instance_path, 'optiq.db')
    app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{db_path}"
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

    db.init_app(app)
    migrate = Migrate(app, db)

    # Enregistrement des blueprints existants
    from Code.routes.activities import activities_bp
    app.register_blueprint(activities_bp)
    from Code.routes.tools import tools_bp
    app.register_blueprint(tools_bp)
    from Code.routes.skills import skills_bp
    app.register_blueprint(skills_bp)
    from Code.routes.softskills import softskills_bp
    app.register_blueprint(softskills_bp)
    from Code.routes.roles import roles_bp
    app.register_blueprint(roles_bp)

    # Enregistrement du blueprint pour la vue des rôles
    from Code.routes.roles_view import roles_view_bp
    app.register_blueprint(roles_view_bp)

    @app.route('/')
    def home():
        return "Bienvenue sur mon application Flask !"

    @app.route('/test_skills')
    def test_skills():
        return app.send_static_file('test_skills.html')

    return app

app = create_app()

if __name__ == '__main__':
    app.run(debug=True)
```

base_sql.py

```
import os
import sys
from flask import Flask
from sqlalchemy import inspect

# Détection du répertoire actuel et du projet
current_dir = os.path.dirname(os.path.abspath(__file__))
project_root = os.path.dirname(current_dir)  # Ajustement ici

# Ajout des chemins au sys.path
sys.path.insert(0, os.path.join(project_root, "Code"))
sys.path.insert(0, project_root)

print("DEBUG: Structure des chemins sys.path :")
for path in sys.path:
    print(f"  - {path}")

# Vérification des fichiers critiques
critical_files = [
    {"name": "extensions.py", "path": os.path.join(project_root, "Code", "extensions.py")},
    {"name": "models.py", "path": os.path.join(project_root, "Code", "models", "models.py")}
]

print("DEBUG: Vérification des fichiers critiques...")
for file in critical_files:
    exists = os.path.exists(file["path"])
    print(f"  - {file['name']} : {'Présent' if exists else 'Manquant'} (Path : {file['path']})")

if any(not os.path.exists(file["path"]) for file in critical_files):
    print("ERREUR : Fichiers critiques manquants.")
    sys.exit(1)

# Tentative d'import des modules
try:
    from extensions import db
    from models.models import Activity, Data, Connection
    print("DEBUG: Import des modules réussi.")
except ImportError as e:
    print(f"ERREUR : Import des modules échoué : {e}")
    sys.exit(1)
```

```
def create_app():
    app = Flask(__name__)
    app.config['SQLALCHEMY_DATABASE_URI'] = f"sqlite:///{os.path.join(project_root,
'Code', 'instance', 'optiq.db')}"
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    db.init_app(app)
    return app

def inspect_tables():
    """Inspecte la structure des tables dans la base."""
    inspector = inspect(db.engine)
    for table_name in inspector.get_table_names():
        print(f"Table: {table_name}")
        for column in inspector.get_columns(table_name):
            print(f"  Column: {column['name']} ({column['type']})")

if __name__ == "__main__":
    app = create_app()
    with app.app_context():
        inspect_tables()
try:
    from models.models import Activity, Data, Connection
    print("DEBUG : Import réussi pour Activity, Data, Connection.")
except ImportError as e:
    print(f"ERREUR : Import échoué : {e}")
    sys.exit(1)
```

extensions.py

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

__init__.py

```
# Code Module initialisé
```

models.py

```
from Code.extensions import db

# Table d'association entre Task et Tool
task_tools = db.Table(
    'task_tools',
    db.Column('task_id', db.Integer, db.ForeignKey('tasks.id'), primary_key=True),
    db.Column('tool_id', db.Integer, db.ForeignKey('tools.id'), primary_key=True)
)

class Activities(db.Model):
    __tablename__ = 'activities'

    id = db.Column(db.Integer, primary_key=True)
    # Identifiant stable provenant de Visio (pour éviter de recréer/supprimer l'activité à
chaque fois)
    shape_id = db.Column(db.String(50), unique=True, index=True, nullable=True)
    name = db.Column(db.String(200), nullable=False)
    description = db.Column(db.Text, nullable=True)
    is_result = db.Column(db.Boolean, nullable=False, default=False)

    # Relation avec les tâches (ordonnées par 'order')
    tasks = db.relationship(
        'Task',
        backref='activity',
        lazy=True,
        order_by='Task.order',
        cascade="all, delete-orphan"
    )

    # Relation avec les compétences validées
    competencies = db.relationship(
        'Competency',
        backref='activity',
        lazy=True,
        cascade="all, delete-orphan"
    )

    # Relation avec les habiletés socio-cognitives (softskills)
    softskills = db.relationship(
        'Softskill',
        backref='activity',
        lazy=True,
        cascade="all, delete-orphan"
    )

class Data(db.Model):
    __tablename__ = 'data'
    id = db.Column(db.Integer, primary_key=True)
    # Permet de faire des mises à jour partielles sur les Data
    shape_id = db.Column(db.String(50), unique=True, index=True, nullable=True)
    name = db.Column(db.String(255), nullable=False)
    type = db.Column(db.String(50), nullable=False)
    description = db.Column(db.Text, nullable=True)
    layer = db.Column(db.String(50), nullable=True)

class Task(db.Model):
    __tablename__ = 'tasks'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255), nullable=False)
    description = db.Column(db.Text, nullable=True)
    order = db.Column(db.Integer, nullable=True)
    activity_id = db.Column(db.Integer, db.ForeignKey('activities.id'), nullable=False)

    # Relation Many-to-Many avec Tool
    tools = db.relationship(
        'Tool',
        secondary=task_tools,
        lazy='subquery',
        backref=db.backref('tasks', lazy=True)
    )

class Tool(db.Model):
    __tablename__ = 'tools'

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255), nullable=False, unique=True)
    description = db.Column(db.Text, nullable=True)

class Competency(db.Model):
    __tablename__ = 'competencies'

    id = db.Column(db.Integer, primary_key=True)
    description = db.Column(db.Text, nullable=False)
    activity_id = db.Column(db.Integer, db.ForeignKey('activities.id'), nullable=False)

class Softskill(db.Model):
    __tablename__ = 'softskills'
    id = db.Column(db.Integer, primary_key=True)
    habilete = db.Column(db.String(255), nullable=False)
    # Stocke le niveau sous forme de chaîne ("1", "2", "3" ou "4")
    niveau = db.Column(db.String(10), nullable=False)
    activity_id = db.Column(db.Integer, db.ForeignKey('activities.id'), nullable=False)

# --- Gestion des rôles ---
class Role(db.Model):
    __tablename__ = 'roles'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), unique=True, nullable=False)

    def __repr__(self):
        return f"<Role {self.name}>"

# Table d'association pour les rôles affectés aux activités.
activity_roles = db.Table('activity_roles',
    db.Column('activity_id', db.Integer, db.ForeignKey('activities.id'), primary_key=True),
    db.Column('role_id', db.Integer, db.ForeignKey('roles.id'), primary_key=True),
    db.Column('status', db.String(50), nullable=False)  # Par exemple "Garant"
)

# Table d'association pour les rôles affectés aux tâches.
task_roles = db.Table('task_roles',
    db.Column('task_id', db.Integer, db.ForeignKey('tasks.id'), primary_key=True),
    db.Column('role_id', db.Integer, db.ForeignKey('roles.id'), primary_key=True),
    db.Column('status', db.String(50), nullable=False)  # Par exemple "Réalisateur",
"Approbateur", etc.
)

# --- Nouvelle table pour les liaisons (Links) ---
class Link(db.Model):
    __tablename__ = 'links'
    id = db.Column(db.Integer, primary_key=True)
    # Si la source est une activité
    source_activity_id = db.Column(db.Integer, db.ForeignKey('activities.id'), nullable=True)
    # Si la source est un Data
    source_data_id = db.Column(db.Integer, db.ForeignKey('data.id'), nullable=True)
    # Si la cible est une activité
    target_activity_id = db.Column(db.Integer, db.ForeignKey('activities.id'), nullable=True)
    # Si la cible est un Data
    target_data_id = db.Column(db.Integer, db.ForeignKey('data.id'), nullable=True)
    type = db.Column(db.String(50), nullable=False)    # Ex: "nourrissante", "déclenchante",
"Retour", etc.
    description = db.Column(db.Text, nullable=True)

    @property
    def source_id(self):
        return self.source_activity_id if self.source_activity_id is not None else
self.source_data_id

    @property
    def target_id(self):
        return self.target_activity_id if self.target_activity_id is not None else self.target_data_id
```

__init__.py

```
# Code/models/__init__.py
from .models import (
    Activities,
    Data,
    Link,
    Task,
    Tool,
    Competency,
    Softskill,
    Role,
    task_tools,
    activity_roles,
    task_roles
)
```

activities.py

```
# Code/routes/activities.py

import os
import io
import contextlib
from flask import Blueprint, jsonify, request, render_template
from sqlalchemy import text  # <-- pour la requête brute du Garant
from Code.extensions import db
from Code.models.models import Activities, Data, Link, Task, Tool, Competency, Softskill
from Code.scripts.extract_visio import process_visio_file, print_summary

activities_bp = Blueprint('activities', __name__, url_prefix='/activities',
template_folder='templates')
```

```
def resolve_return_activity_name(data_record):
    if data_record and data_record.type and data_record.type.lower() == 'retour':
        act = Activities.query.filter_by(name=data_record.name).first()
        if act:
            return act.name
    if data_record and data_record.name:
        return data_record.name
    return "[Nom non renseigné]"

def resolve_data_name_for_incoming(link):
    if link.type and link.type.lower() == 'input':
        data_record = Data.query.get(link.source_id)
        if data_record:
            return resolve_return_activity_name(data_record)
    if link.description:
        return link.description
    return "[Nom non renseigné]"

def resolve_data_name_for_outgoing(link):
    if link.type and link.type.lower() == 'output':
        data_record = Data.query.get(link.target_id)
        if data_record:
            return resolve_return_activity_name(data_record)
    if link.description:
        return link.description
    return "[Nom non renseigné]"

def resolve_activity_name(record_id):
    act = Activities.query.get(record_id)
    if act:
        return act.name
    data_record = Data.query.get(record_id)
    if data_record:
        if data_record.type and data_record.type.lower() == 'retour':
            linked_act = Activities.query.filter_by(name=data_record.name).first()
            if linked_act:
                return linked_act.name
        if data_record.name:
            return data_record.name
    return "[Activité inconnue]"

# AJOUT MINIMAL : récupérer le Garant
```

```
def get_garant_role(activity_id):
    """
    Retourne le rôle Garant associé à l'activité (status='Garant'), ou None.
    """
    result = db.session.execute(
        text("SELECT r.id, r.name FROM activity_roles ar JOIN roles r ON ar.role_id = r.id "
             "WHERE ar.activity_id = :aid AND ar.status = 'Garant'"),
        {"aid": activity_id}
    ).fetchone()
    if result:
        return {"id": result[0], "name": result[1]}
    return None

@activities_bp.route('/', methods=['GET'])
def get_activities():
    try:
        activities = Activities.query.all()
        data = []
        for a in activities:
            data.append({
                "id": a.id,
                "name": a.name,
                "description": a.description or ""
            })
        return jsonify(data), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@activities_bp.route('/', methods=['POST'])
def create_activity():
    data = request.get_json()
    if not data or 'name' not in data:
        return jsonify({"error": "Invalid input. 'name' is required."}), 400
    try:
        new_activity = Activities(name=data['name'], description=data.get('description'))
        db.session.add(new_activity)
        db.session.commit()
        return jsonify({
            "id": new_activity.id,
            "name": new_activity.name,
            "description": new_activity.description or ""
        }), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@activities_bp.route('/<int:activity_id>/tasks/add', methods=['POST'])
def add_task_to_activity(activity_id):
    data = request.get_json()
    if not data or 'name' not in data:
        return jsonify({"error": "Invalid input. 'name' is required."}), 400
    try:
        new_task = Task(
            name=data['name'],
            description=data.get('description', ""),
            activity_id=activity_id
        )
        db.session.add(new_task)
        db.session.commit()
        return jsonify({
            "id": new_task.id,
            "name": new_task.name,
            "description": new_task.description or ""
        }), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@activities_bp.route('/<int:activity_id>/tasks/<int:task_id>', methods=['DELETE'])
def delete_task(activity_id, task_id):
    task = Task.query.filter_by(id=task_id, activity_id=activity_id).first()
    if not task:
        return jsonify({"error": "Task not found for this activity"}), 404
    try:
        db.session.delete(task)
        db.session.commit()
        return jsonify({"message": "Task deleted"}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@activities_bp.route('/<int:activity_id>/tasks/<int:task_id>', methods=['PUT'])
def update_task(activity_id, task_id):
    data = request.get_json()
    if not data or 'name' not in data:
        return jsonify({"error": "Invalid input. 'name' is required."}), 400
    task = Task.query.filter_by(id=task_id, activity_id=activity_id).first()
    if not task:
        return jsonify({"error": "Task not found for this activity"}), 404
    try:
        task.name = data['name']
        task.description = data.get('description', "")
        db.session.commit()
        return jsonify({
            "id": task.id,
            "name": task.name,
            "description": task.description or ""
        }), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@activities_bp.route('/tasks/<int:task_id>/tools/<int:tool_id>', methods=['DELETE'])
def delete_tool_from_task(task_id, tool_id):
    task = Task.query.get(task_id)
    if not task:
        return jsonify({"error": "Task not found"}), 404
    tool = None
    for t in task.tools:
        if t.id == tool_id:
            tool = t
            break
    if not tool:
        return jsonify({"error": "Tool not associated with task"}), 404
    try:
        task.tools.remove(tool)
        db.session.commit()
        return jsonify({"message": "Tool removed from task"}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@activities_bp.route('/<int:activity_id>/tasks/reorder', methods=['POST'])
def reorder_tasks(activity_id):
    data = request.get_json()
    new_order = data.get('order')
    if not new_order:
        return jsonify({"error": "order list is required"}), 400
    try:
        for idx, t_id in enumerate(new_order):
            task = Task.query.filter_by(id=t_id, activity_id=activity_id).first()
            if task:
                task.order = idx
        db.session.commit()
        return jsonify({"message": "Order updated"}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@activities_bp.route('/<int:activity_id>/details', methods=['GET'])
def get_activity_details(activity_id):
    activity = Activities.query.get(activity_id)
    if not activity:
        return jsonify({"error": "Activité non trouvée"}), 404

    tasks_list = []
    tools_list = []
    for t in activity.tasks:
        tasks_list.append(t.name or "")
        for tool in t.tools:
            if tool.name not in tools_list:
                tools_list.append(tool.name)

    input_data_value = getattr(activity, "input_data", "Aucune donnée d'entrée")
    output_data_value = getattr(activity, "output_data", "Aucune donnée de sortie")
    competencies = [{"id": comp.id, "description": comp.description} for comp in
activity.competencies]
    softskills = [{"id": ss.id, "habilete": ss.habilete, "niveau": ss.niveau} for ss in
activity.softskills]

    activity_data = {
        "id": activity.id,
        "name": activity.name,
        "description": activity.description or "",
        "input_data": input_data_value,
        "output_data": output_data_value,
        "tasks": tasks_list,
        "tools": tools_list,
        "competencies": competencies,
        "softskills": softskills
    }
    return jsonify(activity_data), 200
```

```
@activities_bp.route('/update-cartography', methods=['GET'])
def update_cartography():
    try:
        vsdx_path = os.path.join("Code", "example.vsdx")
        process_visio_file(vsdx_path)
        summary_output = io.StringIO()
        with contextlib.redirect_stdout(summary_output):
            print_summary()
        summary_text = summary_output.getvalue()
        return jsonify({
            "message": "Cartographie mise à jour (partielle)",
            "summary": summary_text
        }), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500

@activities_bp.route('/view', methods=['GET'])
def view_activities():
    try:
        # Code original : afficher uniquement les activités non marquées comme résultat
        activities = Activities.query.filter_by(is_result=False).all()

        activity_data = []
        for activity in activities:
            # Connexions entrantes
            incoming_links = Link.query.filter(
                (Link.target_activity_id == activity.id) | (Link.target_data_id == activity.id)
            ).all()
            incoming_list = []
            for link in incoming_links:
                data_name = resolve_data_name_for_incoming(link)
                source_name = resolve_activity_name(link.source_id)
                incoming_list.append({
                    'type': link.type,
                    'data_name': data_name,
                    'source_name': source_name
                })

            # Connexions sortantes
            outgoing_links = Link.query.filter(
                (Link.source_activity_id == activity.id) | (Link.source_data_id == activity.id)
            ).all()
            outgoing_list = []
            for link in outgoing_links:
                data_name = resolve_data_name_for_outgoing(link)
                target_name = resolve_activity_name(link.target_id)
                outgoing_list.append({
                    'type': link.type,
                    'data_name': data_name,
                    'target_name': target_name
                })

            # Tâches
            tasks = sorted(activity.tasks, key=lambda x: x.order if x.order is not None else 0)
            tasks_list = []
            for t in tasks:
                tasks_list.append({
                    'id': t.id,
                    'name': t.name,
                    'description': t.description,
                    'order': t.order,
                    'tools': [
                        {'id': tool.id, 'name': tool.name, 'description': tool.description}
                        for tool in t.tools
                    ]
                })

            # AJOUT MINIMAL : récupérer le Garant
            garant = get_garant_role(activity.id)

            # On ajoute 'garant' dans la data
            activity_data.append({
                'activity': activity,
                'incoming': incoming_list,
                'outgoing': outgoing_list,
                'tasks': tasks_list,
                'garant': garant  # <- nouveau
            })

        return render_template('display_list.html', activity_data=activity_data)
    except Exception as e:
        return f"Erreur lors de l'affichage des activités: {e}", 500

def print_summary():
    print("\n--- RÉSUMÉ DES LIENS ---")
    if link_summaries:
        for (data_name, data_type, s_name, t_name) in link_summaries:
            print(f"  - '{data_name}' ({data_type}) : {s_name} -> {t_name}")
    else:
        print("  Aucun lien créé")
    print("--- Fin du résumé ---\n")
    if rename_summaries:
        print("--- Renommages détectés ---")
        for (old, new) in rename_summaries:
            print(f"  * '{old}' => '{new}'")
        print("--- Fin des renommages ---\n")
    print("CONFIRMATION : toutes les opérations ont été effectuées avec succès.")
```

roles.py

```
# Code/routes/roles.py

from flask import Blueprint, request, jsonify
from sqlalchemy import text
from Code.extensions import db
from Code.models.models import Role

roles_bp = Blueprint('roles', __name__, url_prefix='/roles')

@roles_bp.route('/list', methods=['GET'])
def list_roles():
    """
    Retourne la liste de tous les rôles, triés par ordre alphabétique.
    """
    roles = Role.query.order_by(Role.name).all()
    data = [{"id": r.id, "name": r.name} for r in roles]
    return jsonify(data), 200

@roles_bp.route('/garant/activity/<int:activity_id>', methods=['POST'])
def set_garant_role(activity_id):
    """
    Affecte un rôle Garant à une activité.
    JSON attendu : { "role_name": "<str>" }
    - Si le rôle n'existe pas, on le crée
    - On supprime l'ancien Garant (s'il existe)
    - On insère le nouveau (status='Garant')
    """
    data = request.get_json() or {}
    role_name = data.get("role_name", "").strip()
    if not role_name:
        return jsonify({"error": "role_name is required"}), 400

    # Vérifier si le rôle existe déjà
    existing = Role.query.filter_by(name=role_name).first()
    if not existing:
        existing = Role(name=role_name)
        db.session.add(existing)
        db.session.commit()

    # Supprimer l'ancien Garant
    db.session.execute(
        text("DELETE FROM activity_roles WHERE activity_id=:aid AND status='Garant'"),
        {"aid": activity_id}
    )
    # Ajouter le nouveau
    db.session.execute(
        text("INSERT INTO activity_roles (activity_id, role_id, status) VALUES (:aid, :rid,
'Garant')"),
        {"aid": activity_id, "rid": existing.id}
    )
    db.session.commit()

    return jsonify({
        "message": f"Rôle Garant '{existing.name}' assigné à l'activité {activity_id}",
        "role": {"id": existing.id, "name": existing.name}
    }), 200
```

roles_view.py

```
# Code/routes/roles_view.py

from flask import Blueprint, render_template
from sqlalchemy import text
from Code.extensions import db
from Code.models.models import Role

roles_view_bp = Blueprint('roles_view', __name__, url_prefix='/roles_view',
template_folder='templates')

@roles_view_bp.route('/', methods=['GET'])
def view_roles():
    # Récupérer tous les rôles par ordre alphabétique
    roles = Role.query.order_by(Role.name).all()
```

```
  roles_data = []
  for role in roles:
    # Bloc 1 : Activités où le rôle est Garant
    garant_activities = db.session.execute(
      text("SELECT a.id, a.name, a.description FROM activity_roles ar JOIN activities a ON
ar.activity_id = a.id "
         "WHERE ar.role_id = :rid AND ar.status = 'Garant'"),
      {"rid": role.id}
    ).fetchall()
    block1 = [{"id": row[0], "name": row[1], "description": row[2]} for row in
garant_activities]

    # Bloc 2 : Pour l'instant, on laisse ce bloc vide
    block2 = []

    # Bloc 3 : Compétences associées aux activités où le rôle est Garant
    competencies = db.session.execute(
      text("SELECT c.id, c.description FROM activity_roles ar JOIN competencies c ON
ar.activity_id = c.activity_id "
         "WHERE ar.role_id = :rid AND ar.status = 'Garant'"),
      {"rid": role.id}
    ).fetchall()
    block3 = [{"id": comp[0], "description": comp[1]} for comp in competencies]

    # Bloc 4 : Habiletés socio-cognitives associées aux activités où le rôle est Garant
    softskills = db.session.execute(
      text("SELECT s.habilete, s.niveau FROM activity_roles ar JOIN softskills s ON
ar.activity_id = s.activity_id "
         "WHERE ar.role_id = :rid AND ar.status = 'Garant'"),
      {"rid": role.id}
    ).fetchall()
    hsc_dict = {}
    for habilete, niveau in softskills:
      try:
        niveau_int = int(niveau)
      except:
        niveau_int = 0
      if habilete in hsc_dict:
        if niveau_int > hsc_dict[habilete]:
          hsc_dict[habilete] = niveau_int
      else:
        hsc_dict[habilete] = niveau_int
        block4 = [{"habilete": k, "niveau": str(v)} for k, v in hsc_dict.items()]

        roles_data.append({
            "role": {"id": role.id, "name": role.name},
            "block1": block1,
            "block2": block2,
            "block3": block3,
            "block4": block4
        })
    return render_template("roles_view.html", roles_data=roles_data)
```

skills.py

```
import os
import openai
import json
from flask import Blueprint, request, jsonify
from Code.extensions import db
from Code.models.models import Competency

skills_bp = Blueprint('skills', __name__, url_prefix='/skills')

@skills_bp.route('/propose', methods=['POST'])
def propose_skills():
    """
    Génère 2 ou 3 propositions de compétences via l'IA (NF X50-124).
    Reçoit en JSON les infos de l'activité (name, input_data, output_data, tasks, tools).
    """
    data = request.get_json() or {}
    activity_name = data.get("name", "Activité sans nom")
    input_data_value = data.get("input_data", "")
    output_data_value = data.get("output_data", "")

    # Extraire la liste de tâches
    tasks_data = data.get("tasks", [])
    if tasks_data and isinstance(tasks_data[0], dict):
        tasks_list = [t.get("name", "") for t in tasks_data]
    else:
        tasks_list = tasks_data if isinstance(tasks_data, list) else []
    tasks_str = ", ".join(tasks_list) if tasks_list else ""

    # Extraire la liste d'outils
    tools_data = data.get("tools", [])
    if tools_data and isinstance(tools_data[0], dict):
        tools_list = [t.get("name", "") for t in tools_data]
    else:
        tools_list = tools_data if isinstance(tools_data, list) else []
    tools_str = ", ".join(tools_list) if tools_list else ""

    prompt = f"""
Vous êtes un expert en gestion des compétences selon la norme NF X50-124.
Proposez 2 ou 3 phrases de compétences pour l'activité suivante :

- Nom : {activity_name}
- Données d'entrée : {input_data_value}
- Données de sortie : {output_data_value}
- Tâches : {tasks_str or "Aucune tâche"}
- Outils : {tools_str or "Aucun outil"}

Contraintes :
1) Chaque phrase doit être rédigée en une seule phrase sans utiliser de listes.
2) Ne mentionnez pas l'environnement ni le niveau de performance.
3) Ne listez pas explicitement "Données, Tâches, Outils".
4) Chaque phrase doit commencer par un verbe d'action.
5) Générez exactement 2 ou 3 phrases, chacune sur une nouvelle ligne.
"""

    openai.api_key = os.getenv("OPENAI_API_KEY")
    if not openai.api_key:
        return jsonify({"error": "Clé OpenAI manquante (OPENAI_API_KEY)."}), 500

    try:
        response = openai.ChatCompletion.create(
            model="gpt-4",
            messages=[
                {"role": "system", "content": "Vous êtes un assistant spécialisé en compétences NF
X50-124."},
                {"role": "user", "content": prompt}
            ],
            temperature=0.3,
            max_tokens=400
        )
        raw_text = response['choices'][0]['message']['content'].strip()
        lines = [l.strip() for l in raw_text.split("\n") if l.strip()]
        return jsonify({"proposals": lines}), 200
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

```
@skills_bp.route('/add', methods=['POST'])
def add_competency():
    """
    Ajoute une compétence dans la table 'competencies'.
    JSON attendu : { "activity_id": <int>, "description": <str> }
    """
    data = request.get_json() or {}
    activity_id = data.get("activity_id")
    description = data.get("description", "").strip()
    if not activity_id or not description:
        return jsonify({"error": "activity_id and description are required"}), 400

    comp = Competency(activity_id=activity_id, description=description)
    try:
        db.session.add(comp)
        db.session.commit()
        return jsonify({
            "id": comp.id,
            "activity_id": comp.activity_id,
            "description": comp.description
        }), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@skills_bp.route('/<int:competency_id>', methods=['PUT'])
def update_competency(competency_id):
    """
    Met à jour une compétence existante.
    JSON attendu : { "description": <str> }
    """
    data = request.get_json() or {}
    new_desc = data.get("description", "").strip()
    if not new_desc:
        return jsonify({"error": "description is required"}), 400

    comp = Competency.query.get(competency_id)
    if not comp:
        return jsonify({"error": "Competency not found"}), 404

    try:
        comp.description = new_desc
        db.session.commit()
        return jsonify({
            "id": comp.id,
            "description": comp.description
        }), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@skills_bp.route('/<int:competency_id>', methods=['DELETE'])
def delete_competency(competency_id):
    comp = Competency.query.get(competency_id)
    if not comp:
        return jsonify({"error": "Competency not found"}), 404
    try:
        db.session.delete(comp)
        db.session.commit()
        return jsonify({"message": "Competency deleted"}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500
```

softskills.py

```
import os
import openai
import json
from flask import Blueprint, request, jsonify
from Code.extensions import db
from Code.models.models import Softskill

softskills_bp = Blueprint('softskills_bp', __name__, url_prefix='/softskills')

@softskills_bp.route('/propose', methods=['POST'])
def propose_softskills():
    """
    Propose 3-4 habiletés socio-cognitives (HSC) via l'IA, renvoyées en JSON.
    """
    data = request.get_json()
    if not data:
        return jsonify({"error": "Aucune donnée reçue"}), 400

    activity_info = data.get("activity", "")
    competencies_info = data.get("competencies", "")
    x50_766_hsc = """
Les habiletés socio-cognitives officielles de la norme X50-766 sont :
Relation à soi :
 - Auto-évaluation
 - Auto-régulation
 - Auto-organisation
 - Auto-mobilisation
Relation à l’autre :
 - Sensibilité sociale
 - Adaptation relationnelle
 - Coopération
Relation à l’action :
 - Raisonnement logique
 - Planification
 - Arbitrage
Relation au savoir :
 - Traitement de l’information
 - Synthèse
 - Conceptualisation
Relation à la complexité :
 - Flexibilité mentale
 - Projection
 - Approche globale
"""

    prompt = f"""
Voici une activité avec ses compétences existantes :

Activité : {activity_info}
Compétences existantes : {competencies_info}

Propose 3 ou 4 habiletés socio-cognitives officielles (norme X50-766) jugées essentielles
pour cette activité.
Utilise uniquement la liste suivante (n'invente pas d'autres habiletés) :
{x50_766_hsc}

Pour chaque habileté, indique un niveau entre 1 et 4 (1 = Aptitude, 4 = Excellence).
Réponds au format JSON, par exemple :
[
  {{"habilete": "Auto-évaluation", "niveau": "2"}},
  {{"habilete": "Planification", "niveau": "3"}}
]
```

```
"""
    openai.api_key = os.getenv("OPENAI_API_KEY")
    if not openai.api_key:
        return jsonify({"error": "Clé OpenAI manquante (OPENAI_API_KEY)."}), 500
    try:
        response = openai.ChatCompletion.create(
            model="gpt-4",
            messages=[
                {"role": "system", "content": "Tu es un expert en habiletés socio-cognitives X50-
766."},
                {"role": "user", "content": prompt}
            ],
            temperature=0.2,
            max_tokens=400
        )
        ai_message = response.choices[0].message['content'].strip()
        proposals = json.loads(ai_message)
        return jsonify(proposals)
    except Exception as e:
        return jsonify({"error": f"Erreur lors de la récupération des habiletés socio-cognitives :
{str(e)}"}), 500

@softskills_bp.route('/add', methods=['POST'])
def add_softskill():
    """
    Enregistre une softskill (HSC) dans la table 'softskills'.
    JSON attendu : { "activity_id": <int>, "habilete": <str>, "niveau": <str> }
    """
    data = request.get_json() or {}
    activity_id = data.get("activity_id")
    habilete = data.get("habilete", "").strip()
    niveau = data.get("niveau", "").strip()
    if not activity_id or not habilete or not niveau:
        return jsonify({"error": "activity_id, habilete and niveau are required"}), 400
    new_softskill = Softskill(activity_id=activity_id, habilete=habilete, niveau=niveau)
    try:
        db.session.add(new_softskill)
        db.session.commit()
        return jsonify({
            "id": new_softskill.id,
            "activity_id": new_softskill.activity_id,
            "habilete": new_softskill.habilete,
            "niveau": new_softskill.niveau
        }), 201
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@softskills_bp.route('/translate', methods=['POST'])
def translate_softskills():
    """
    Reçoit un texte libre (user_input) et renvoie une liste d'HSC.
    Pour limiter la réponse à 3 à 5 HSC, le prompt demande explicitement de ne pas proposer
plus de 5 objets.
    """
    data = request.get_json() or {}
    user_input = data.get("user_input", "").strip()
    if not user_input:
        return jsonify({"error": "Aucune donnée reçue pour la traduction."}), 400

    x50_766_hsc = """
Les habiletés socio-cognitives officielles de la norme X50-766 sont :
Relation à soi :
 - Auto-évaluation
 - Auto-régulation
 - Auto-organisation
 - Auto-mobilisation
Relation à l'autre :
 - Sensibilité sociale
 - Adaptation relationnelle
 - Coopération
Relation à l'action :
 - Raisonnement logique
 - Planification
 - Arbitrage
Relation au savoir :
 - Traitement de l'information
 - Synthèse
 - Conceptualisation
Relation à la complexité :
 - Flexibilité mentale
 - Projection
 - Approche globale
"""

    prompt = f"""
```

```
Voici un texte libre décrivant des soft skills :
"{user_input}"

Analyse ce texte dans le contexte de l'activité et des tâches associées, et traduis-le en une
liste de 3 à 5 habiletés socio-cognitives issues de la norme X50-766.
Utilise uniquement la liste suivante (n'invente pas d'autres habiletés) :
{x50_766_hsc}

Pour chaque habileté, attribue un niveau entre 1 et 4 (1 = Aptitude, 4 = Excellence).
Réponds au format JSON, par exemple :
[
 {{"habilete": "Auto-évaluation", "niveau": "2"}},
 {{"habilete": "Planification", "niveau": "3"}}
]

Ne propose jamais plus de 5 objets dans le tableau.
"""

    openai.api_key = os.getenv("OPENAI_API_KEY")
    if not openai.api_key:
        return jsonify({"error": "Clé OpenAI manquante (OPENAI_API_KEY)."}), 500

    try:
        response = openai.ChatCompletion.create(
            model="gpt-4",
            messages=[
                {"role": "system", "content": "Tu es un expert en habiletés socio-cognitives selon la
norme X50-766."},
                {"role": "user", "content": prompt}
            ],
            temperature=0.2,
            max_tokens=400
        )
        ai_message = response.choices[0].message['content'].strip()
        proposals = json.loads(ai_message)
        return jsonify({"proposals": proposals})
    except Exception as e:
        return jsonify({"error": f"Erreur lors de la traduction des softskills : {str(e)}"}), 500

@softskills_bp.route('/<int:softskill_id>', methods=['PUT'])
def update_softskill(softskill_id):
    data = request.get_json() or {}
    new_habilete = data.get("habilete", "").strip()
    new_niveau = data.get("niveau", "").strip()
    if not new_habilete or not new_niveau:
        return jsonify({"error": "habilete and niveau are required"}), 400

    ss = Softskill.query.get(softskill_id)
    if not ss:
        return jsonify({"error": "Softskill not found"}), 404

    try:
        ss.habilete = new_habilete
        ss.niveau = new_niveau
        db.session.commit()
        return jsonify({
            "id": ss.id,
            "habilete": ss.habilete,
            "niveau": ss.niveau
        }), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500

@softskills_bp.route('/<int:softskill_id>', methods=['DELETE'])
def delete_softskill(softskill_id):
    ss = Softskill.query.get(softskill_id)
    if not ss:
        return jsonify({"error": "Softskill not found"}), 404
    try:
        db.session.delete(ss)
        db.session.commit()
        return jsonify({"message": "Softskill deleted"}), 200
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500
```

tasks.py

```
from flask import Blueprint, request, jsonify
from Code.extensions import db
from Code.models.models import Task, Activities

tasks_bp = Blueprint('tasks', __name__, url_prefix='/tasks')

@tasks_bp.route('/add', methods=['POST'])
def add_task():
    """
    Ajoute une tâche associée à une activité.
    Expects JSON with keys: activity_id, name, (optionnellement description et order).
    """
    data = request.get_json()
    if not data or 'activity_id' not in data or 'name' not in data:
        return jsonify({'error': 'Données invalides. "activity_id" et "name" sont requis.'}), 400

    activity = Activities.query.get(data['activity_id'])
    if not activity:
        return jsonify({'error': 'Activité non trouvée.'}), 404

    new_task = Task(
        name=data['name'],
        description=data.get('description', ''),
        order=data.get('order', None),
        activity_id=data['activity_id']
    )
    db.session.add(new_task)
    db.session.commit()

    return jsonify({
        'id': new_task.id,
        'name': new_task.name,
        'description': new_task.description,
        'order': new_task.order,
        'activity_id': new_task.activity_id
    }), 201
```

tools.py

```
from flask import Blueprint, request, jsonify
from Code.extensions import db
from Code.models.models import Task, Tool

tools_bp = Blueprint('tools', __name__, url_prefix='/tools')

@tools_bp.route('/add', methods=['POST'])
def add_tools_to_task():
    data = request.get_json()
    if not data or 'task_id' not in data:
        return jsonify({"error": "task_id is required"}), 400
    task = Task.query.get(data['task_id'])
    if not task:
        return jsonify({"error": "Task not found"}), 404
    added_tools = []
    try:
        if 'existing_tool_ids' in data and isinstance(data['existing_tool_ids'], list):
            for tool_id in data['existing_tool_ids']:
                tool = Tool.query.get(tool_id)
                if tool and tool not in task.tools:
                    task.tools.append(tool)
                    added_tools.append({"id": tool.id, "name": tool.name})
        if 'new_tools' in data and isinstance(data['new_tools'], list):
            for tool_name in data['new_tools']:
                if tool_name:
                    tool = Tool.query.filter(db.func.lower(Tool.name) == tool_name.lower()).first()
                    if not tool:
                        tool = Tool(name=tool_name)
                        db.session.add(tool)
                        db.session.flush()  # obtenir l'id
                    if tool not in task.tools:
                        task.tools.append(tool)
                        added_tools.append({"id": tool.id, "name": tool.name})
        db.session.commit()
    except Exception as e:
        db.session.rollback()
        return jsonify({"error": str(e)}), 500
    return jsonify({"task_id": task.id, "added_tools": added_tools}), 200

@tools_bp.route('/all', methods=['GET'])
def get_all_tools():
    tools = Tool.query.all()
    return jsonify([{'id': tool.id, 'name': tool.name} for tool in tools])
```

ui_routes.py

```
from flask import Blueprint, render_template
from Code.models.models import Activities

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 = Activities.query.all()
    return render_template('ui/activities.html', activities=activities)
```

__init__.py

```
# Initialisation du module routes
```

activities_list.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Liste des activités</title>
  <!-- Font Awesome pour les icônes -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.0/css/all.min.css">
  <!-- SortableJS pour le drag & drop -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/Sortable/1.15.0/Sortable.min.js"></script>
  <style>
    body { font-family: Arial, sans-serif; }
    .update-btn {
      background-color: green;
      color: white;
      padding: 10px 15px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
      margin-bottom: 20px;
      font-size: 1em;
    }
    .activity-container { border: 1px solid #aaa; margin-bottom: 20px; overflow: hidden; }
    .activity-header {
      background-color: #add8e6;
      padding: 5px;
      font-size: 0.9em;
      cursor: pointer;
      display: flex;
      align-items: center;
    }
    .activity-header h2 { margin: 0; flex-grow: 1; }
    .toggle-icon { font-size: 1em; margin-right: 5px; }
    .activity-details { padding: 5px; display: none; }
    .connections-container {
      display: flex;
      justify-content: space-between;
      flex-wrap: wrap;
      margin-top: 10px;
    }
    .connections-container > div { width: 48%; }
    .conn-table { width: 100%; border-collapse: collapse; margin-bottom: 10px; }
    .conn-table th, .conn-table td { border: 1px solid #ccc; padding: 4px; text-align: left; }
    .conn-table th { background-color: #cce5ff; }
    /* Mise en forme conditionnelle des données */
    .declenchante { font-weight: bold; }
    .nourrissante { font-style: italic; }
    .tasks-section { margin-top: 10px; }
    .task {
      border: 1px solid #ddd;
      padding: 5px;
      margin-bottom: 10px;
      display: flex;
      flex-direction: column;
    }
    /* Ligne principale de la tâche en deux colonnes */
    .task-row {
      display: flex;
      justify-content: space-between;
      align-items: flex-start;
    }
    .task-left {
      display: flex;
      align-items: center;
      gap: 5px;
      flex: 1;
    }
    .task-right {
      display: flex;
      flex-direction: column;
      align-items: flex-start;
      gap: 5px;
      max-width: 250px;
    }
    .task-title { max-width: 45ch; word-wrap: break-word; }
    /* Liste des outils : chaque li sur sa propre ligne */
    .tools-list ul {
      list-style: none;
      padding: 0;
      margin: 0;
    }
```

```
    .tools-list li {
      background: #f0f0f0;
      padding: 2px 5px;
      border-radius: 3px;
      margin-bottom: 5px;
      display: block;
    }
    /* Bouton d'ajout d'outil : affiché dans un li à part */
    .add-tool-li {
      background: none;
      border: none;
      padding: 0;
      margin: 0;
      display: block;
    }
    .edit-task-form {
      display: none;
      margin-top: 5px;
      border: 1px solid #ccc;
      padding: 5px;
      width: 100%;
    }
    .task-form, .tool-form {
      margin-top: 5px;
      border: 1px solid #ccc;
      padding: 5px;
    }
    .icon-btn {
      background: none;
      border: none;
      cursor: pointer;
      margin-left: 5px;
    }
    .icon-btn i { font-size: 1em; }
  </style>
</head>
<body>
  <button id="update-cartography-button" class="update-btn">
    <i class="fa-solid fa-arrows-rotate"></i> Mettre à jour la cartographie
  </button>
  <h1>Liste des activités</h1>
  <!-- Ligne de test pour vérifier que ce template est bien utilisé -->
  <p style="color:red; font-weight:bold;">Test : {{ test_hello }}</p>
```

```
{% for item in activity_data %}
 <div class="activity-container">
  <div class="activity-header" onclick="toggleDetails('details-{{ item.activity.id }}', this)">
   <span class="toggle-icon" id="icon-{{ item.activity.id }}">▶</span>
   <h2>{{ item.activity.name }}</h2>
  </div>
  <div class="activity-details" id="details-{{ item.activity.id }}">
   <p>{{ item.activity.description or "" }}</p>
   <div class="connections-container">
    <div class="incoming-connections">
     <h3>Connexions entrantes</h3>
     {% if item.incoming and item.incoming|length > 0 %}
      <table class="conn-table">
       <tr>
        <th>Nom de la donnée</th>
        <th>Provenance</th>
       </tr>
       {% for conn in item.incoming %}
        <tr>
         <td>
          {% if conn.type|lower == 'déclenchante' %}
           <span class="declenchante">{{ conn.data_name }}</span>
          {% elif conn.type|lower == 'nourrissante' %}
           <span class="nourrissante">{{ conn.data_name }}</span>
          {% else %}
           {{ conn.data_name }}
          {% endif %}
         </td>
         <td>{{ conn.source_name }}</td>
        </tr>
       {% endfor %}
      </table>
     {% else %}
      <p>Aucune connexion entrante.</p>
     {% endif %}
    </div>
    <div class="outgoing-connections">
     <h3>Connexions sortantes</h3>
     {% if item.outgoing and item.outgoing|length > 0 %}
      <table class="conn-table">
       <tr>
        <th>Nom de la donnée</th>
        <th>Vers</th>
```

```
            </tr>
            {% for conn in item.outgoing %}
              <tr>
                <td>
                  {% if conn.type|lower == 'déclenchante' %}
                    <span class="declenchante">{{ conn.data_name }}</span>
                  {% elif conn.type|lower == 'nourrissante' %}
                    <span class="nourrissante">{{ conn.data_name }}</span>
                  {% else %}
                    {{ conn.data_name }}
                  {% endif %}
                </td>
                <td>{{ conn.target_name }}</td>
              </tr>
            {% endfor %}
          </table>
        {% else %}
          <p>Aucune connexion sortante.</p>
        {% endif %}
      </div>
    </div>
    <!-- Section des tâches -->
    <div class="tasks-section">
      <h3>Tâches</h3>
      <div id="tasks-container-{{ item.activity.id }}">
        {% if item.tasks and item.tasks|length > 0 %}
          <ul id="tasks-list-{{ item.activity.id }}">
            {% for task in item.tasks %}
              <li class="task" id="task-{{ task.id }}" data-task-id="{{ task.id }}">
                <div class="task-row">
                  <!-- Colonne de gauche -->
                  <div class="task-left">
                    <i class="fa-solid fa-bars icon-btn" style="cursor: move;"></i>
                    <span class="task-title">
                      <strong id="task-name-display-{{ task.id }}">{{ task.name }}</strong>
                      {% if task.description %}
                        - <span id="task-desc-display-{{ task.id }}">{{ task.description }}</span>
                      {% endif %}
                    </span>
                    <button class="icon-btn" onclick="deleteTask('{{ item.activity.id }}', '{{ task.id
}}')">
                      <i class="fa-solid fa-trash"></i>
                    </button>
                    <button class="icon-btn" onclick="showEditTaskForm('{{ item.activity.id }}', '{{
task.id }}', '{{ task.name }}', '{{ task.description | default('') }}')">
                      <i class="fa-solid fa-pencil"></i>
                    </button>
                  </div>
                  <!-- Colonne de droite -->
                  <div class="task-right">
                    <div class="tools-list" id="tools-for-task-{{ task.id }}">
                      <ul>
                        {% if task.tools and task.tools|length > 0 %}
                          {% for tool in task.tools %}
                            <li data-tool-id="{{ tool.id }}">
                              <span class="tool-name">{{ tool.name }}</span>
                              <button class="icon-btn" onclick="deleteToolFromTask('{{ task.id }}', '{{
tool.id }}')">
                            <i class="fa-solid fa-trash"></i>
                              </button>
                            </li>
                          {% endfor %}
                          <!-- Bouton "+" dans un li séparé placé après la liste des outils -->
                          <li class="add-tool-li">
                            <button class="icon-btn" onclick="showToolForm('{{ task.id }}')">
                              <i class="fa-solid fa-plus"></i>
                            </button>
                          </li>
                        {% else %}
                          <li id="no-tools-msg-{{ task.id }}">
                            Aucun outil associé.
                            <button class="icon-btn" onclick="showToolForm('{{ task.id }}')">
                              <i class="fa-solid fa-plus"></i>
                            </button>
                          </li>
                        {% endif %}
                      </ul>
                    </div>
                  </div>
                </div>
                <!-- Formulaire d'édition de la tâche, affiché sous la ligne -->
                <div class="edit-task-form" id="edit-task-form-{{ task.id }}">
                  <input type="text" id="edit-task-name-{{ task.id }}" placeholder="Nom de la
tâche" />
                  <input type="text" id="edit-task-desc-{{ task.id }}" placeholder="Description
(optionnelle)" />
```

```
            <button onclick="submitEditTask('{{ item.activity.id }}', '{{ task.id
}}')">Enregistrer</button>
            <button onclick="hideEditTaskForm('{{ task.id }}')">Annuler</button>
          </div>
          <!-- Formulaire d'ajout d'outils (invisible par défaut) -->
          <div id="tool-form-{{ task.id }}" class="tool-form" style="display: none;">
            <div>
              <label for="existing-tools-{{ task.id }}">Outils existants:</label>
              <select id="existing-tools-{{ task.id }}" multiple style="width:
100%;"></select>
            </div>
            <div>
              <label for="new-tools-{{ task.id }}">Nouveaux outils (séparés par des
virgules):</label>
              <input type="text" id="new-tools-{{ task.id }}" placeholder="Ex: Outil1, Outil2"
style="width: 100%;" />
            </div>
            <button onclick="submitTools('{{ task.id }}')">Enregistrer</button>
            <button onclick="hideToolForm('{{ task.id }}')">Annuler</button>
          </div>
        </li>
      {% endfor %}
    </ul>
  {% else %}
    <p id="no-tasks-message-{{ item.activity.id }}">Aucune tâche enregistrée.</p>
  {% endif %}
 </div>
 <button onclick="showTaskForm('{{ item.activity.id }}')">
  <i class="fa-solid fa-plus"></i> Ajouter une tâche
 </button>
 <div id="task-form-{{ item.activity.id }}" class="task-form" style="display:none;">
  <input type="text" id="task-name-{{ item.activity.id }}" placeholder="Nom de la
tâche" />
  <input type="text" id="task-desc-{{ item.activity.id }}" placeholder="Description
(optionnelle)" />
  <button onclick="submitTask('{{ item.activity.id }}')">Enregistrer</button>
  <button onclick="hideTaskForm('{{ item.activity.id }}')">Annuler</button>
 </div>
</div>
<!-- Bouton pour proposer des compétences pour cette activité -->
<button onclick="fetchActivityDetailsForSkills({{ item.activity.id }})" style="margin-
top:10px; padding:8px 12px; font-size:0.9em;">
 Proposer Compétences
```

```
        </button>
      </div>
    </div>
  {% endfor %}
  <script>
    // Bouton de mise à jour de la cartographie
    document.getElementById('update-cartography-button').addEventListener('click',
function() {
        fetch('/activities/update-cartography')
        .then(function(response) { return response.json(); })
        .then(function(data) {
          alert(data.message + "\n" + data.summary);
          location.reload();
        })
        .catch(function(error) {
          alert("Erreur lors de la mise à jour de la cartographie: " + error.message);
        });
    });
    function toggleDetails(detailsId, headerElem) {
        var detailsElem = document.getElementById(detailsId);
        var iconElem = headerElem.querySelector('.toggle-icon');
        var currentDisplay = window.getComputedStyle(detailsElem).display;
        if (currentDisplay === "none") {
          detailsElem.style.display = "block";
          iconElem.textContent = "▼";
        } else {
          detailsElem.style.display = "none";
          iconElem.textContent = "▶";
        }
    }
    // Gestion du formulaire d'ajout de tâche
    function showTaskForm(activityId) {
        document.getElementById('task-form-' + activityId).style.display = 'block';
    }
    function hideTaskForm(activityId) {
        document.getElementById('task-form-' + activityId).style.display = 'none';
    }
    function submitTask(activityId) {
        var taskName = document.getElementById('task-name-' + activityId).value;
        var taskDesc = document.getElementById('task-desc-' + activityId).value;
        if (!taskName) {
          alert("Le nom de la tâche est requis.");
          return;
        }
        fetch('/activities/' + activityId + '/tasks/add', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ name: taskName, description: taskDesc })
        })
        .then(function(response) {
          if (response.ok) return response.json();
          throw new Error("Erreur lors de l'ajout de la tâche.");
        })
        .then(function(data) {
          var tasksList = document.getElementById('tasks-list-' + activityId);
          if (!tasksList) {
            var noTasksMsg = document.getElementById('no-tasks-message-' + activityId);
            if (noTasksMsg) { noTasksMsg.parentNode.removeChild(noTasksMsg); }
            tasksList = document.createElement('ul');
            tasksList.id = 'tasks-list-' + activityId;
            var tasksSection = document.getElementById('task-form-' + activityId).parentNode;
            tasksSection.appendChild(tasksList);
          }
          var li = document.createElement('li');
          li.className = 'task';
          li.id = 'task-' + data.id;
          li.setAttribute('data-task-id', data.id);
          li.innerHTML = '<div class="task-row">' +
                  '<div class="task-left">' +
                  '<i class="fa-solid fa-bars icon-btn" style="cursor: move;"></i>' +
                  '<span class="task-title"><strong id="task-name-display-' + data.id + '">' +
data.name + '</strong>' +
                  (data.description ? ' - <span id="task-desc-display-' + data.id + '">' +
data.description + '</span>' : '') +
                  '</span>' +
                  '<button class="icon-btn" onclick="deleteTask(\'' + activityId + '\', \'' +
data.id + '\')"><i class="fa-solid fa-trash"></i></button>' +
                  '<button class="icon-btn" onclick="showEditTaskForm(\'' + activityId + '\', \''
+ data.id + '\', \'' + data.name + '\', \'' + data.description + '\')"><i class="fa-solid fa-
pencil"></i></button>' +
                  '</div>' +
                  '<div class="task-right">' +
                  '<div class="tools-list" id="tools-for-task-' + data.id + '">' +
                  '<ul>' +
                  '<li id="no-tools-msg-' + data.id + '">Aucun outil associé.</li>' +
                  '<li class="add-tool-li">' +
                  '<button class="icon-btn" onclick="showToolForm(\'' + data.id + '\')"><i
class="fa-solid fa-plus"></i></button>' +
                  '</li>' +
                  '</ul>' +
                  '</div>' +
                  '</div>' +
                  '</div>' +
                  '<div class="edit-task-form" id="edit-task-form-' + data.id + '">' +
                  '<input type="text" id="edit-task-name-' + data.id + '" placeholder="Nom de la
tâche" />' +
                  '<input type="text" id="edit-task-desc-' + data.id + '"
placeholder="Description (optionnelle)" />' +
                  '<button onclick="submitEditTask(\'' + activityId + '\', \'' + data.id +
'\')">Enregistrer</button>' +
                  '<button onclick="hideEditTaskForm(\'' + data.id + '\')">Annuler</button>' +
                  '</div>' +
                  '<div id="tool-form-' + data.id + '" class="tool-form" style="display: none;">' +
                  '<div><label for="existing-tools-' + data.id + '">Outils existants:</label>' +
                  '<select id="existing-tools-' + data.id + '" multiple style="width:
100%;"></select></div>' +
                  '<div><label for="new-tools-' + data.id + '">Nouveaux outils (séparés par des
virgules):</label>' +
                  '<input type="text" id="new-tools-' + data.id + '" placeholder="Ex: Outil1,
Outil2" style="width: 100%;" /></div>' +
                  '<button onclick="submitTools(\'' + data.id + '\')">Enregistrer</button>' +
                  '<button onclick="hideToolForm(\'' + data.id + '\')">Annuler</button>' +
                  '</div>';
          tasksList.appendChild(li);
          document.getElementById('task-name-' + activityId).value = "";
          document.getElementById('task-desc-' + activityId).value = "";
          hideTaskForm(activityId);
        })
        .catch(function(error) {
          alert(error.message);
        });
      }
      // Suppression d'une tâche
      function deleteTask(activityId, taskId) {
        if (!confirm("Confirmez-vous la suppression de cette tâche ?")) return;
        fetch('/activities/' + activityId + '/tasks/' + taskId, { method: 'DELETE' })
        .then(function(response) {
          if (response.ok) return response.json();
          throw new Error("Erreur lors de la suppression de la tâche.");
```

```
        })
        .then(function(data) {
            alert(data.message);
            var taskElem = document.getElementById('task-' + taskId);
            if (taskElem) { taskElem.parentNode.removeChild(taskElem); }
        })
        .catch(function(error) {
            alert(error.message);
        });
    }
    // Fonction pour afficher le formulaire d'édition d'une tâche
    function showEditTaskForm(activityId, taskId, name, description) {
        document.getElementById('edit-task-form-' + taskId).style.display = 'block';
        document.getElementById('edit-task-name-' + taskId).value = name;
        document.getElementById('edit-task-desc-' + taskId).value = description;
    }
    function hideEditTaskForm(taskId) {
        document.getElementById('edit-task-form-' + taskId).style.display = 'none';
    }
    function submitEditTask(activityId, taskId) {
        var newName = document.getElementById('edit-task-name-' + taskId).value;
        var newDesc = document.getElementById('edit-task-desc-' + taskId).value;
        if (!newName) {
            alert("Le nom de la tâche est requis.");
            return;
        }
        fetch('/activities/' + activityId + '/tasks/' + taskId, {
            method: 'PUT',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ name: newName, description: newDesc })
        }).then(function(response) {
            if (response.ok) return response.json();
            throw new Error("Erreur lors de la mise à jour de la tâche.");
        }).then(function(data) {
            var nameElem = document.getElementById('task-name-display-' + taskId);
            if (nameElem) { nameElem.textContent = data.name; }
            var descElem = document.getElementById('task-desc-display-' + taskId);
            if (descElem) {
                descElem.textContent = data.description;
            } else {
                var taskTitle = document.querySelector('#task-name-display-' +
taskId).parentNode;
                var span = document.createElement('span');
            span.id = 'task-desc-display-' + taskId;
            span.textContent = data.description;
            taskTitle.appendChild(document.createTextNode(" - "));
            taskTitle.appendChild(span);
          }
          hideEditTaskForm(taskId);
        }).catch(function(error) {
          alert(error.message);
        });
      }
      // Suppression d'un outil d'une tâche sans recharger la page
      function deleteToolFromTask(taskId, toolId) {
        if (!confirm("Confirmez-vous la suppression de cet outil de la tâche ?")) return;
        fetch('/activities/tasks/' + taskId + '/tools/' + toolId, { method: 'DELETE' })
        .then(function(response) {
          if (response.ok) return response.json();
          throw new Error("Erreur lors de la suppression de l'outil.");
        })
        .then(function(data) {
          var toolsContainer = document.getElementById('tools-for-task-' + taskId);
          var ul = toolsContainer.querySelector('ul');
          if (!ul) return;
          var toolElem = ul.querySelector('li[data-tool-id="' + toolId + '"]');
          if (toolElem) {
            toolElem.parentNode.removeChild(toolElem);
          }
          // Vérifier si le bouton "+" existe ; si non, le réinsérer
          var addBtn = ul.querySelector('li.add-tool-li');
          if (!addBtn) {
            var newAddLi = document.createElement('li');
            newAddLi.className = 'add-tool-li';
            newAddLi.innerHTML = '<button class="icon-btn" onclick="showToolForm(\'' +
taskId + '\')"><i class="fa-solid fa-plus"></i></button>';
            ul.appendChild(newAddLi);
          }
        })
        .catch(function(error) {
          alert(error.message);
        });
      }
      // Fonctions pour gérer le formulaire d'ajout d'outils
      function showToolForm(taskId) {
        document.getElementById('tool-form-' + taskId).style.display = 'block';
        fetch('/tools/all')
        .then(function(response) { return response.json(); })
        .then(function(data) {
            var selectElem = document.getElementById('existing-tools-' + taskId);
            selectElem.innerHTML = "";
            data.forEach(function(tool) {
                var option = document.createElement('option');
                option.value = tool.id;
                option.text = tool.name;
                selectElem.appendChild(option);
            });
        })
        .catch(function(error) {
            console.error("Erreur lors du chargement des outils existants:", error);
        });
    }
    function hideToolForm(taskId) {
        document.getElementById('tool-form-' + taskId).style.display = 'none';
    }
    function submitTools(taskId) {
        var selectElem = document.getElementById('existing-tools-' + taskId);
        var newToolsInput = document.getElementById('new-tools-' + taskId);
        var existingToolIds = Array.from(selectElem.selectedOptions).map(function(option) {
return parseInt(option.value); });
        var newTools = newToolsInput.value.split(",").map(function(item) { return item.trim();
}).filter(function(item) { return item.length > 0; });
        var payload = {
            task_id: parseInt(taskId),
            existing_tool_ids: existingToolIds,
            new_tools: newTools
        };
        fetch('/tools/add', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(payload)
        })
        .then(function(response) {
            if (response.ok) return response.json();
            return response.json().then(function(err) { throw new Error(err.error || "Erreur lors
de l'ajout des outils."); });
        })
        .then(function(data) {
            var toolsContainer = document.getElementById('tools-for-task-' + taskId);
            var ul = toolsContainer.querySelector('ul');
            if (!ul) {
              ul = document.createElement('ul');
              toolsContainer.appendChild(ul);
            }
            // Retirer le bouton "+" existant s'il est présent
            var addBtn = ul.querySelector('li.add-tool-li');
            if (addBtn) {
              addBtn.parentNode.removeChild(addBtn);
            }
            data.added_tools.forEach(function(tool) {
              var li = document.createElement('li');
              li.setAttribute("data-tool-id", tool.id);
              li.innerHTML = tool.name + ' <button class="icon-btn"
onclick="deleteToolFromTask(\'' + taskId + '\', \'' + tool.id + '\')"><i class="fa-solid fa-
trash"></i></button>';
              ul.appendChild(li);
            });
            // Réinsérer le bouton "+" en tant que li isolé
            var newAddLi = document.createElement('li');
            newAddLi.className = 'add-tool-li';
            newAddLi.innerHTML = '<button class="icon-btn" onclick="showToolForm(\'' +
taskId + '\')"><i class="fa-solid fa-plus"></i></button>';
            ul.appendChild(newAddLi);
            selectElem.selectedIndex = -1;
            newToolsInput.value = "";
            hideToolForm(taskId);
          })
          .catch(function(error) {
            alert(error.message);
          });
      }
      // Initialisation du drag & drop pour réordonner les tâches et sauvegarder l'ordre
      document.addEventListener('DOMContentLoaded', function() {
        var taskLists = document.querySelectorAll('[id^="tasks-list-"]');
        taskLists.forEach(function(list) {
          Sortable.create(list, {
            animation: 150,
            handle: '.fa-bars',
            onEnd: function(evt) {
              var list = evt.from;
              var listId = list.getAttribute('id'); // e.g. tasks-list-123
              var activityId = listId.split('-')[2]; // Extrait l'id de l'activité
              var newOrder = [];
              list.querySelectorAll('li.task').forEach(function(taskElem) {
                newOrder.push(taskElem.getAttribute('data-task-id'));
              });
              fetch('/activities/' + activityId + '/tasks/reorder', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ order: newOrder })
              }).then(function(response) {
                if (!response.ok) {
                  console.error("Erreur de sauvegarde de l'ordre");
                }
              });
            }
          });
        });
      });
  </script>
  {% include "skills_section.html" %}
</body>
</html>
```

activity_competencies.html

```
<h3>Compétences</h3>

<ul id="competencies-list-{{ item.activity.id }}">
  {% for comp in item.activity.competencies %}
    <li data-comp-id="{{ comp.id }}" style="margin-bottom:5px;">
      <span class="validated-skill-text">{{ comp.description }}</span>
      <button onclick="editCompetency(this, {{ comp.id }})" style="margin-left:5px;">
        <i class="fa-solid fa-pencil"></i>
      </button>
      <button onclick="deleteCompetency(this, {{ comp.id }})" style="margin-left:5px;">
        <i class="fa-solid fa-trash"></i>
      </button>
      <!-- Formulaire d'édition local (caché) -->
      <div class="edit-competency-form" id="edit-competency-form-{{ comp.id }}"
style="display:none; margin-top:5px;">
        <label>Description :</label>
        <input type="text" id="edit-competency-desc-{{ comp.id }}" value="{{ comp.description
}}" />
        <button onclick="submitEditCompetency('{{ comp.id }}')">Enregistrer</button>
        <button onclick="hideEditCompetencyForm('{{ comp.id }}')">Annuler</button>
    </div>
   </li>
  {% endfor %}
</ul>

<!-- Bouton pour proposer des compétences (ouvre le modal via skills_section.html) -->
<button onclick="fetchActivityDetailsForSkills({{ item.activity.id }})">
  Proposer Compétences
</button>
```

activity_connections.html

```
<p>{{ item.activity.description or "" }}</p>
<div class="connections-container">
  <div class="incoming-connections">
    <h3>Connexions entrantes</h3>
    {% if item.incoming and item.incoming|length > 0 %}
      <table class="conn-table">
        <tr>
          <th>Nom de la donnée</th>
          <th>Provenance</th>
        </tr>
        {% for conn in item.incoming %}
          <tr>
            <td>
              {% if conn.type|lower == 'déclenchante' %}
                <span class="declenchante">{{ conn.data_name }}</span>
              {% elif conn.type|lower == 'nourrissante' %}
                <span class="nourrissante">{{ conn.data_name }}</span>
              {% else %}
                {{ conn.data_name }}
              {% endif %}
            </td>
            <td>{{ conn.source_name }}</td>
          </tr>
        {% endfor %}
      </table>
    {% else %}
      <p>Aucune connexion entrante.</p>
    {% endif %}
  </div>
  <div class="outgoing-connections">
    <h3>Connexions sortantes</h3>
    {% if item.outgoing and item.outgoing|length > 0 %}
      <table class="conn-table">
        <tr>
          <th>Nom de la donnée</th>
          <th>Vers</th>
        </tr>
        {% for conn in item.outgoing %}
          <tr>
            <td>
              {% if conn.type|lower == 'déclenchante' %}
                <span class="declenchante">{{ conn.data_name }}</span>
              {% elif conn.type|lower == 'nourrissante' %}
                <span class="nourrissante">{{ conn.data_name }}</span>
              {% else %}
                {{ conn.data_name }}
              {% endif %}
            </td>
            <td>{{ conn.target_name }}</td>
          </tr>
        {% endfor %}
      </table>
    {% else %}
      <p>Aucune connexion sortante.</p>
    {% endif %}
  </div>
</div>
```

activity_header.html

```
<div class="activity-container">
  <div class="activity-header" onclick="toggleDetails('details-{{ item.activity.id }}', this)">
    <span class="toggle-icon" id="icon-{{ item.activity.id }}">▶</span>
    <h2>{{ item.activity.name }}</h2>
  </div>
  <div class="activity-details" id="details-{{ item.activity.id }}">
```

activity_softskills.html

```
<h3>Habiletés socio-cognitives</h3>

<div id="softskills-list-{{ activity_id }}">
 {% for ss in item.activity.softskills %}
  <div class="softskill-item" style="margin-bottom:5px;" data-ss-id="{{ ss.id }}">
    <span class="softskill-text">{{ ss.habilete }} (Niveau: <span class="softskill-level">{{
ss.niveau }}</span>)</span>
```

```
      <i class="fas fa-pencil-alt edit-softskill" title="Modifier"></i>
      <i class="fas fa-trash delete-softskill" title="Supprimer"></i>
      <div class="edit-softskill-form" id="edit-softskill-form-{{ ss.id }}" style="display:none;">
        <label>Habileté :</label>
        <input type="text" id="edit-softskill-name-{{ ss.id }}" value="{{ ss.habilete }}" />
        <label>Niveau (1..4) :</label>
        <input type="number" min="1" max="4" id="edit-softskill-level-{{ ss.id }}" value="{{
ss.niveau }}" />
        <button onclick="submitEditSoftskillFromDOM('{{ ss.id }}')">Enregistrer</button>
        <button onclick="hideEditSoftskillForm('{{ ss.id }}')">Annuler</button>
      </div>
    </div>
  {% endfor %}
</div>

<button class="define-hsc-btn" data-activity-id="{{ activity_id }}">
  Proposer HSC
</button>
<button onclick="openTranslateSoftskillsModal({{ activity_id }})">
  Traduire Softskills
</button>

<div class="softskill-form">
  <label for="softskill-name-{{ activity_id }}">Nom de l'HSC :</label>
  <input type="text" id="softskill-name-{{ activity_id }}" />

  <label for="softskill-level-{{ activity_id }}">Niveau (1..4) :</label>
  <input type="number" min="1" max="4" id="softskill-level-{{ activity_id }}" />

  <button onclick="submitSoftskill('{{ activity_id }}')">Ajouter</button>
</div>
```

activity_tasks.html

```
<!-- activity_tasks.html -->
<div class="tasks-section">
    <h3>Tâches</h3>
    <ul id="tasks-list-{{ item.activity.id }}">
      {% for task in item.tasks %}
        <li class="task" id="task-{{ task.id }}" data-task-id="{{ task.id }}">
          <div class="task-row">
            <div class="task-left">
              <i class="fa-solid fa-bars icon-btn" style="cursor: move;"></i>
              <span class="task-title">
                <strong id="task-name-display-{{ task.id }}">{{ task.name }}</strong>
                {% if task.description %}
                  - <span id="task-desc-display-{{ task.id }}">{{ task.description }}</span>
                {% endif %}
              </span>
              <button class="icon-btn" onclick="deleteTask('{{ item.activity.id }}', '{{ task.id }}')">
                <i class="fa-solid fa-trash"></i>
              </button>
              <button class="icon-btn" onclick="showEditTaskForm('{{ item.activity.id }}', '{{
task.id }}', '{{ task.name }}', '{{ task.description|default('') }}')">
                <i class="fa-solid fa-pencil"></i>
              </button>
            </div>
            <div class="task-right">
              <div class="tools-list" id="tools-for-task-{{ task.id }}">
                <ul>
                  {% if task.tools and task.tools|length > 0 %}
                    {% for tool in task.tools %}
                      <li data-tool-id="{{ tool.id }}">
                        <span class="tool-name">{{ tool.name }}</span>
                        <button class="icon-btn" onclick="deleteToolFromTask('{{ task.id }}', '{{ tool.id
}}')">
                      <i class="fa-solid fa-trash"></i>
                        </button>
                      </li>
                    {% endfor %}
                  {% else %}
                    <li id="no-tools-msg-{{ task.id }}">Aucun outil associé.</li>
                  {% endif %}
                  <li class="add-tool-li">
                    <button class="icon-btn add-tool-btn" onclick="showToolForm('{{ task.id }}')">
                      <i class="fa-solid fa-plus"></i>
                    </button>
                  </li>
                </ul>
              </div>
            </div>
          </div>

          <!-- Formulaire d'édition de la tâche -->
          <div class="edit-task-form" id="edit-task-form-{{ task.id }}">
            <input type="text" id="edit-task-name-{{ task.id }}" placeholder="Nom de la tâche" />
            <input type="text" id="edit-task-desc-{{ task.id }}" placeholder="Description
```

```
(optionnelle)" />
          <button onclick="submitEditTask('{{ item.activity.id }}', '{{ task.id
}}')">Enregistrer</button>
          <button onclick="hideEditTaskForm('{{ task.id }}')">Annuler</button>
        </div>

        <!-- Formulaire d'ajout d'outils (invisible par défaut) -->
        <div id="tool-form-{{ task.id }}" class="tool-form" style="display: none;">
          <div>
            <label for="existing-tools-{{ task.id }}">Outils existants:</label>
            <select id="existing-tools-{{ task.id }}" multiple style="width: 100%;"></select>
          </div>
          <div>
            <label for="new-tools-{{ task.id }}">Nouveaux outils (séparés par des
virgules):</label>
            <input type="text" id="new-tools-{{ task.id }}" placeholder="Ex: Outil1, Outil2"
style="width: 100%;" />
          </div>
          <button onclick="submitTools('{{ task.id }}')">Enregistrer</button>
          <button onclick="hideToolForm('{{ task.id }}')">Annuler</button>
        </div>
      </li>
    {% endfor %}
   </ul>

   <!-- Bouton pour ajouter une nouvelle tâche -->
   <button onclick="showTaskForm('{{ item.activity.id }}')" style="margin-top:5px;
padding:5px 10px; font-size:0.8em;">
     <i class="fa-solid fa-plus"></i> Ajouter une tâche
   </button>

   <!-- Formulaire d'ajout de tâche (invisible par défaut) -->
   <div id="task-form-{{ item.activity.id }}" class="task-form" style="display:none;">
     <input type="text" id="task-name-{{ item.activity.id }}" placeholder="Nom de la tâche"
/>
     <input type="text" id="task-desc-{{ item.activity.id }}" placeholder="Description
(optionnelle)" />
     <button onclick="submitTask('{{ item.activity.id }}')">Enregistrer</button>
     <button onclick="hideTaskForm('{{ item.activity.id }}')">Annuler</button>
   </div>
  </div>
```

competency_modal.html

```
<div id="competencyModal"
    style="display:none; position:fixed; left:20%; top:20%; width:60%; background:#fff;
         border:1px solid #aaa; padding:10px; z-index:9999;">
  <h4>Propositions de compétences :</h4>
  <ul id="proposalsList" style="list-style:none; padding-left:0;"></ul>
  <div style="margin-top:10px;">
    <button onclick="validateAllSelectedProposals()">Valider les compétences</button>
    <button onclick="closeCompetencyModal()">Annuler</button>
  </div>
</div>

<script>
let proposalsGlobal = [];
let currentActivityId = null;

function showProposalsModal(proposals, activityId) {
  proposalsGlobal = proposals;
  currentActivityId = activityId;
  const proposalsList = document.getElementById('proposalsList');
  proposalsList.innerHTML = "";

  proposals.forEach((proposal, index) => {
    const li = document.createElement('li');
    li.style.marginBottom = "5px";
    li.innerHTML = `
      <input type="checkbox" id="proposal-${index}" data-proposal="${proposal}">
      <label for="proposal-${index}">${proposal}</label>
    `;
    proposalsList.appendChild(li);
  });

  document.getElementById('competencyModal').style.display = "block";
}

function validateAllSelectedProposals() {
  const proposalsList = document.getElementById('proposalsList');
  const checkboxes = proposalsList.querySelectorAll('input[type="checkbox"]:checked');
  if (checkboxes.length === 0) {
    alert("Veuillez sélectionner au moins une proposition.");
    return;
  }
  let addPromises = [];
  checkboxes.forEach(cb => {
    const text = cb.getAttribute('data-proposal') || "Compétence ?";
    let p = fetch('/skills/add', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        activity_id: currentActivityId,
        description: text
      })
    })
    .then(response => response.json())
    .then(data => {
      if (data.error) {
        console.error("Erreur en ajoutant la compétence:", data.error);
      } else {
        // Ajouter dans le DOM
        addCompetencyItemToDOM(currentActivityId, data.id, data.description);
      }
    })
    .catch(error => {
      console.error("Erreur /skills/add:", error);
    });
    addPromises.push(p);
  });
  Promise.all(addPromises).then(() => {
    closeCompetencyModal();
    alert("Compétences sauvegardées en base.");
  });
}

function closeCompetencyModal() {
  document.getElementById('competencyModal').style.display = "none";
}
</script>
```

display_list.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Liste des activités</title>

  <!-- Font Awesome pour les icônes -->
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.0/css/all.min.css">

  <!-- Feuille de style principale -->
  <link rel="stylesheet" href="/static/optiq.css">

  <!-- SortableJS pour le drag & drop -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/Sortable/1.15.0/Sortable.min.js"></script>

  <!-- jQuery nécessaire pour les fonctions -->
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
  <!-- Bouton pour mettre à jour la cartographie -->
  <button id="update-cartography-button" class="update-btn">
    <i class="fa-solid fa-arrows-rotate"></i> Mettre à jour la cartographie
  </button>

  <!-- Bouton pour accéder à la vue des rôles -->
  <a href="/roles_view/" style="display:inline-block; margin:10px 0; padding:10px 15px;
background:#007BFF; color:#fff; text-decoration:none; border-radius:5px;">
    Voir la vue des rôles
  </a>

  <h1>Liste des activités</h1>

  {% for item in activity_data %}
  <div class="activity-container">
    <!-- En-tête cliquable -->
    <div class="activity-header" onclick="toggleDetails('details-{{ item.activity.id }}', this)">
      <span class="toggle-icon" id="icon-{{ item.activity.id }}">▶</span>
      <h2>{{ item.activity.name }}</h2>
    </div>

    <!-- Bloc pour afficher le rôle Garant -->
    <div style="margin:5px 0;">
      <span id="activity-garant-{{ item.activity.id }}">
        Garant : {% if item.garant %}{{ item.garant.name }}{% else %}Aucun{% endif %}
      </span>
      <button onclick="openGarantModal({{ item.activity.id }})" style="margin-
left:10px;">Changer Garant</button>
    </div>
    <!-- Zone de détails (initialement masquée) -->
    <div class="activity-details" id="details-{{ item.activity.id }}" style="display:none;">
      <p>{{ item.activity.description or "" }}</p>

      <!-- Connexions entrantes / sortantes -->
      {% include "activity_connections.html" %}

      <!-- Tâches -->
      {% include "activity_tasks.html" %}

      <!-- Compétences validées + bouton "Proposer Compétences" -->
      {% include "activity_competencies.html" %}

      {% set activity_id = item.activity.id %}
      <!-- Habiletés socio-cognitives -->
      {% include "activity_softskills.html" %}
    </div>
  </div>
  {% endfor %}

  <!-- Inclusion du modal pour le rôle Garant -->
  {% include "roles_modal.html" %}

  <!-- Inclusion des modules pour compétences & softskills -->
  {% include "translate_softskills_modal.html" %}
  {% include "skills_section.html" %}
  {% include "competency_modal.html" %}

  <!-- Inclusion des fichiers JavaScript -->
  <script src="/static/js/tasks.js"></script>
  <script src="/static/js/tools.js"></script>
  <script src="/static/js/competencies.js"></script>
  <script src="/static/js/softskills.js"></script>
  <script src="/static/js/translate_softskills.js"></script>
  <script src="/static/js/main.js"></script>
  <script src="/static/js/roles.js"></script>

  <script>
    document.getElementById('update-cartography-button').addEventListener('click',
function() {
      fetch('/activities/update-cartography')
      .then(response => response.json())
      .then(data => {
        let msg = (data.message || "") + "\n" + (data.summary || "");
        alert(msg.trim());
        location.reload();
      })
      .catch(error => {
        alert("Erreur lors de la mise à jour de la cartographie: " + error.message);
      });
    });

    function toggleDetails(detailsId, headerElem) {
      var detailsElem = document.getElementById(detailsId);
      var iconElem = headerElem.querySelector('.toggle-icon');
      var currentDisplay = window.getComputedStyle(detailsElem).display;
      if (currentDisplay === "none") {
        detailsElem.style.display = "block";
        iconElem.textContent = "▼";
      } else {
        detailsElem.style.display = "none";
        iconElem.textContent = "▶";
      }
    }
  </script>
</body>
</html>
```

roles_modal.html

```
<!-- Code/routes/templates/roles_modal.html -->
<div id="garantModal" style="display:none; position: fixed; top:20%; left:30%; width:40%;
background: #fff; border:1px solid #aaa; padding:20px; z-index:9999;">
  <h3>Choisir le rôle Garant</h3>
  <input type="hidden" id="garant-activity-id" value="">
  <label for="garant-role-select">Rôle existant :</label>
  <select id="garant-role-select" style="width:100%;"></select>
  <br/><br/>
  <label for="garant-new-role">Ou saisir un nouveau rôle :</label>
  <input type="text" id="garant-new-role" style="width:100%;" placeholder="Ex: Chef de
projet">
  <br/><br/>
  <button onclick="submitGarantRole()">Valider</button>
  <button onclick="closeGarantModal()">Annuler</button>
 </div>
```

roles_view.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Vue des Rôles</title>
  <link rel="stylesheet" href="/static/optiq.css">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.4.0/css/all.min.css">
  <style>
    /* Conteneur global du rôle */
    .role-container {
      border: 1px solid #ccc;
      padding: 10px;
      margin-bottom: 10px;
      background-color: #2ecc71; /* Fond vert pour le rôle */
      color: #fff;
    }
    .role-header {
      cursor: pointer;
      font-size: 1.2em;
      font-weight: bold;
      padding: 5px;
      background-color: #27ae60;
    }
    .toggle-icon {
      margin-right: 5px;
    }
    /* Blocs internes (le header conserve la couleur d'origine) */
    .role-block {
      margin: 5px 0;
      border: 1px solid rgba(255,255,255,0.6);
    }
    .block-1 { background-color: #1e8449; }
    .block-2 { background-color: #27ae60; }
    .block-3 { background-color: #27ae60; }
    .block-4 { background-color: #a9dfbf; }
    .block-header {
      cursor: pointer;
      font-size: 1em;
      font-weight: bold;
      padding: 3px;
      border-bottom: 1px solid rgba(255,255,255,0.6);
      color: #fff;
    }
    /* Style commun pour le contenu déployé de tous les blocs */
    .block-content {
      display: none;
      background-color: #fff;
      color: #000;
      padding: 10px;
    }
    /* Bouton Retour */
    .back-button {
      display: inline-block;
      margin: 10px 0;
      padding: 10px 15px;
      background-color: #007BFF;
      color: #fff;
      text-decoration: none;
      border-radius: 5px;
    }
  </style>
</head>
<body>
  <!-- Bouton pour revenir à la vue des activités -->
  <a href="/activities/view" class="back-button">
    <i class="fa-solid fa-arrow-left"></i> Retour aux activités
  </a>

  <h1>Vue des Rôles</h1>

  {% for item in roles_data %}
    <div class="role-container">
      <!-- En-tête du rôle, cliquable pour déplier/replier le conteneur complet -->
      <div class="role-header" onclick="toggleRoleContainer('role-body-{{ item.role.id }}',
this)">
        <span class="toggle-icon" id="role-icon-{{ item.role.id }}">▶</span>
        Rôle : {{ item.role.name }}
      </div>
      <div id="role-body-{{ item.role.id }}" style="display:none;">
        <!-- Bloc 1 : Activités où ce rôle est Garant -->
        <div class="role-block block-1">
          <div class="block-header" onclick="toggleBlock('block1-{{ item.role.id }}', this)">
            <span class="toggle-icon" id="block1-icon-{{ item.role.id }}">▶</span>
            Bloc 1 : Activités où ce rôle est Garant
    </div>
    <div id="block1-{{ item.role.id }}" class="block-content">
      {% if item.block1 %}
        <ul>
          {% for act in item.block1 %}
            <li>{{ act.name }} (ID: {{ act.id }}) - {{ act.description }}</li>
          {% endfor %}
        </ul>
      {% else %}
        <p>Aucune activité trouvée.</p>
      {% endif %}
    </div>
  </div>
  <!-- Bloc 2 : Activités/Tâches où ce rôle intervient (non Garant) -->
  <div class="role-block block-2">
    <div class="block-header" onclick="toggleBlock('block2-{{ item.role.id }}', this)">
      <span class="toggle-icon" id="block2-icon-{{ item.role.id }}">▶</span>
      Bloc 2 : Activités/Tâches où ce rôle intervient (non Garant)
    </div>
    <div id="block2-{{ item.role.id }}" class="block-content">
      {% if item.block2 %}
        <ul>
          {% for act in item.block2 %}
            <li>{{ act.name }} (ID: {{ act.id }})</li>
          {% endfor %}
        </ul>
      {% else %}
        <p>Aucune donnée pour ce bloc.</p>
      {% endif %}
    </div>
  </div>
  <!-- Bloc 3 : Compétences associées -->
  <div class="role-block block-3">
    <div class="block-header" onclick="toggleBlock('block3-{{ item.role.id }}', this)">
      <span class="toggle-icon" id="block3-icon-{{ item.role.id }}">▶</span>
      Bloc 3 : Compétences associées aux activités Garant
    </div>
    <div id="block3-{{ item.role.id }}" class="block-content">
      {% if item.block3 %}
        <ul>
          {% for comp in item.block3 %}
            <li>{{ comp.description }} (ID: {{ comp.id }})</li>
          {% endfor %}
          </ul>
        {% else %}
          <p>Aucune compétence trouvée.</p>
        {% endif %}
      </div>
    </div>
    <!-- Bloc 4 : Habiletés socio-cognitives -->
    <div class="role-block block-4">
      <div class="block-header" onclick="toggleBlock('block4-{{ item.role.id }}', this)">
        <span class="toggle-icon" id="block4-icon-{{ item.role.id }}">▶</span>
        Bloc 4 : Habiletés socio-cognitives
      </div>
      <div id="block4-{{ item.role.id }}" class="block-content">
        {% if item.block4 %}
          <ul>
            {% for hsc in item.block4 %}
              <li>{{ hsc.habilete }} - Niveau: {{ hsc.niveau }}</li>
            {% endfor %}
          </ul>
        {% else %}
          <p>Aucune habileté trouvée.</p>
        {% endif %}
      </div>
    </div>
  </div>
</div>
{% endfor %}

<script>
 function toggleRoleContainer(containerId, headerElem) {
   var container = document.getElementById(containerId);
   var roleId = containerId.split('-')[1];
   var icon = document.getElementById('role-icon-' + roleId);
   if (container.style.display === "none") {
     container.style.display = "block";
     icon.textContent = "▼";
   } else {
     container.style.display = "none";
     icon.textContent = "▶";
   }
 }

 function toggleBlock(blockId, headerElem) {
    var block = document.getElementById(blockId);
    var icon = document.getElementById(blockId.replace('block', 'block-icon'));
    if (!icon) {
      icon = headerElem.querySelector('.toggle-icon');
    }
    if (block.style.display === "none") {
      block.style.display = "block";
      if (icon) { icon.textContent = "▼"; }
    } else {
      block.style.display = "none";
      if (icon) { icon.textContent = "▶"; }
    }
  }
 </script>
</body>
</html>
```

skills_section.html

```
<script>
 function fetchActivityDetailsForSkills(activityId) {
   fetch(`/activities/${activityId}/details`)
     .then(response => response.json())
     .then(data => {
       if (data.error) {
         alert("Erreur : " + data.error);
         return;
       }
       proposeSkills(data);
     })
     .catch(error => {
       alert("Erreur lors de la récupération des données : " + error.message);
     });
 }

 function proposeSkills(activityData) {
   fetch('/skills/propose', {
     method: 'POST',
     headers: { 'Content-Type': 'application/json' },
     body: JSON.stringify(activityData)
   })
   .then(response => response.json())
   .then(data => {
     if (data.error) {
       alert("Erreur dans la proposition: " + data.error);
       return;
     }
     showProposalsModal(data.proposals, activityData.id);
   })
   .catch(error => {
     alert("Erreur : " + error.message);
   });
 }
 </script>
```

translate_softskills_modal.html

```
<div id="translateSoftskillsModal"
     style="display:none; position:fixed; left:20%; top:20%; width:60%; background:#fff;
            border:1px solid #aaa; padding:10px; z-index:9999;">
  <h4>Traduire Softskills</h4>
  <p>Entrez ci-dessous ce que vous considérez comme soft skills :</p>
  <textarea id="translateSoftskillsInput" style="width:100%; height:100px;"></textarea>
  <div style="margin-top:10px;">
    <button onclick="submitSoftskillsTranslation()">Valider</button>
    <button onclick="closeTranslateSoftskillsModal()">Annuler</button>
  </div>
</div>

<script>
function openTranslateSoftskillsModal(activityId) {
  console.log("openTranslateSoftskillsModal - activityId:", activityId);
  window.translateSoftskillsActivityId = activityId;
  document.getElementById('translateSoftskillsModal').style.display = 'block';
}

function closeTranslateSoftskillsModal() {
  console.log("closeTranslateSoftskillsModal");
  document.getElementById('translateSoftskillsModal').style.display = 'none';
  window.translateSoftskillsActivityId = null;
}

function submitSoftskillsTranslation() {
  console.log("submitSoftskillsTranslation triggered");
  const activityId = window.translateSoftskillsActivityId;
  if (!activityId) {
    alert("Identifiant de l'activité introuvable.");
    return;
```

```
}
const userInputElem = document.getElementById('translateSoftskillsInput');
const userInput = userInputElem.value.trim();
if (!userInput) {
  alert("Veuillez saisir du texte.");
  return;
}

// Appel /softskills/translate
$.ajax({
  url: '/softskills/translate',
  method: 'POST',
  contentType: 'application/json',
  data: JSON.stringify({ user_input: userInput }),
  success: function(response) {
    console.log("Réponse de /softskills/translate:", response);
    if (!response.proposals) {
      alert("Réponse inattendue : pas de 'proposals' !");
      return;
    }
    let addPromises = [];
    response.proposals.forEach(function(item) {
      let p = $.ajax({
        url: '/softskills/add',
        method: 'POST',
        contentType: 'application/json',
        data: JSON.stringify({
          activity_id: activityId,
          habilete: item.habilete,
          niveau: item.niveau
        }),
        success: function(added) {
          if (added.error) {
            console.error("Erreur ajout HSC:", added.error);
          } else {
            addSoftskillItemToDOM(activityId, added.habilete, added.niveau, added.id);
          }
        },
        error: function(err) {
          console.error("Erreur /softskills/add:", err);
        }
      });
      addPromises.push(p);
    });
    $.when.apply($, addPromises).then(function() {
      userInputElem.value = "";
      closeTranslateSoftskillsModal();
    });
  },
  error: function() {
    alert("Erreur lors de la traduction des softskills.");
  }
 });
}
</script>
```

extract_visio.py

```
import os
import sys
from vsdx import VisioFile

# Pour pouvoir importer Code.extensions et Code.models.models
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..', '..')))

from Code.extensions import db
from Code.models.models import Activities, Data, Link

# Calques Visio gérés
LAYER_MAPPING = {
    "1": "Activity",    # rectangle normal
    "9": "N link",      # ligne pointillée => nourrissante
    "10": "T link",     # ligne pleine => déclenchante
    "6": "Result",      # drapeau => activité de résultat
    "8": "Return"       # rond => retour
}

# Calques à ignorer
IGNORE_LAYERS = ["légende", "Color"]

# Mappings globaux (shape_id => ID en base)
activity_mapping = {}
data_mapping = {}
return_mapping = {}  # retours

# Stocke en mémoire les connecteurs rencontrés : [ { 'data_id':..., 'data_name':..., 'data_type':..., 'from_id':..., 'to_id':... }, ... ]
```

```
connectors_list = []

# Pour résumé / logs
link_summaries = []     # liste (data_name, data_type, source_name, target_name)
rename_summaries = []   # liste (old_name, new_name)


def create_app():
    """Exécuter ce script directement (standalone)."""
    from flask import Flask
    app = Flask(__name__)
    instance_path = os.path.abspath(os.path.join("Code", "instance"))
    if not os.path.exists(instance_path):
        os.makedirs(instance_path)
    db_path = os.path.join(instance_path, "optiq.db")
    app.config["SQLALCHEMY_DATABASE_URI"] = f"sqlite:///{db_path}"
    app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
    db.init_app(app)
    return app


def process_visio_file(vsdx_path):
    """
    1) Parcours toutes les pages / formes
    2) Gère creation / maj / suppression de Activities/Data + fusion retours
    3) Vide la table 'links'
    4) Reconstruit tous les liens (Link) depuis 'connectors_list'
    5) Nettoie orphelins
    6) Affiche un résumé
    """
    if not os.path.exists(vsdx_path):
        print(f"ERREUR : Fichier Visio introuvable : {vsdx_path}")
        return

    print(f"INFO : Démarrage de l’import depuis {vsdx_path}")

    # Réinit
    activity_mapping.clear()
    data_mapping.clear()
    return_mapping.clear()
    connectors_list.clear()
    link_summaries.clear()
    rename_summaries.clear()
    # 1) Parcours du visio
    with VisioFile(vsdx_path) as visio:
        for page in visio.pages:
            print(f"INFO : Analyse de la page : {page.name}")
            for shape in page.all_shapes:
                process_shape(shape)

    # 2) Suppression des activités et data obsolètes
    del_act_count = remove_activities_not_in_new_mapping()
    del_data_count = remove_data_not_in_new_mapping()

    # 3) On vide la table 'links'
    Link.query.delete()
    db.session.commit()
    print("INFO : Table 'links' vidée, on va la reconstruire à partir de connectors_list...")

    # 4) Reconstruire la table des liens
    rebuild_links_from_connectors()

    # 5) Nettoyage final orphelins (optionnel)
    cleanup_orphan_links()

    # Récap
    print("INFO : Import terminé.")
    print(f"     Activités ajoutées/mises à jour : {len(activity_mapping)} ; supprimées :
{del_act_count}")
    print(f"     Data (connecteurs/retours) totaux :
{len(data_mapping)+len(return_mapping)} ; supprimés : {del_data_count}")
    print_summary()


def process_shape(shape):
    """Oriente selon le calque (Activity, Return, T link, N link, etc.)."""
    layer = get_layer(shape)
    if not layer:
        return
    if layer.lower() in [x.lower() for x in IGNORE_LAYERS]:
        return

    if layer == "Activity":
        add_or_update_activity(shape, is_result=False)
    elif layer == "Result":
        add_or_update_activity(shape, is_result=True)
    elif layer == "Return":
        add_or_update_return(shape)
    elif layer in ("T link", "N link"):
        store_connector_info(shape, layer)
    else:
        print(f"INFO : Calque '{layer}' non traité => forme '{shape.text or '??'}' ignorée.")


#################################################################
################
# A) Gestion Activities
#################################################################
################

def add_or_update_activity(shape, is_result=False):
    key = standardize_id(shape.ID)
    txt = shape.text.strip() if shape.text else ("Résultat sans nom" if is_result else "Activité
sans nom")

    fill = get_fill_pattern(shape)
    if fill and fill != "1":
        is_result = True

    act = Activities.query.filter_by(shape_id=key).first()
    if act:
        changed = False
        old_name = act.name
        if old_name != txt:
            act.name = txt
            rename_summaries.append((old_name, txt))
            changed = True
        if act.is_result != is_result:
            act.is_result = is_result
            changed = True
        if changed:
            print(f"INFO : Activity (ID={act.id}) maj => name='{txt}', is_result={is_result}")
        else:
            print(f"INFO : Activity (ID={act.id}) déjà existante, pas de modif.")
    else:
        new_a = Activities(name=txt, is_result=is_result, shape_id=key)
        db.session.add(new_a)
        db.session.flush()
        print(f"INFO : Activity créée => '{txt}' (ID={new_a.id}, shape_id={key}, 
is_result={is_result})")
        act = new_a

    activity_mapping[key] = act.id
    db.session.commit()


def remove_activities_not_in_new_mapping():
    existing_acts = Activities.query.filter(Activities.shape_id.isnot(None)).all()
    count = 0
    for act in existing_acts:
        if act.shape_id not in activity_mapping:
            print(f"INFO : Suppression Activity '{act.name}' (ID={act.id}, 
shape_id={act.shape_id})")
            db.session.delete(act)
            count += 1
    db.session.commit()
    return count


##############################################################
################
# B) Gestion Data / Retours
##############################################################
################

def add_or_update_return(shape):
    """
    Gère une forme 'Return' => Data(type='Retour').
    1) Crée/MAJ
    2) Fusion si d'autres Retours ont le même nom
    """
    key = standardize_id(shape.ID)
    txt = shape.text.strip() or "Retour sans nom"

    d = Data.query.filter_by(shape_id=key, type="Retour").first()
    if d:
        old = d.name
        if old != txt:
            d.name = txt
            rename_summaries.append((old, txt))
            db.session.commit()
            print(f"INFO : Return (ID={d.id}) renommé '{old}' => '{txt}'")
    else:
        d = Data(name=txt, type="Retour", shape_id=key)
        db.session.add(d)
        db.session.flush()
        print(f"INFO : Return créé => '{txt}' (ID={d.id}, shape_id={key})")

    return_mapping[key] = d.id
    db.session.commit()

    # Fusion
    unify_retours(d)


def unify_retours(d):
    """Si d’autres Data(type='Retour') ont le même .name, on supprime (sauf d)."""
    keeper_id = d.id
    duplicates = Data.query.filter(
        Data.type == "Retour",
        Data.name == d.name,
        Data.id != keeper_id
    ).all()
    for dupe in duplicates:
        print(f"INFO : Fusion retours => supprime Return ID={dupe.id}
shape_id={dupe.shape_id}")
        db.session.delete(dupe)
    db.session.commit()


def store_connector_info(shape, layer):
    """
    Au lieu de créer direct le link, on stocke en mémoire :
    - data (type déclenchante/nourrissante)
    - shape_id => data en base
    - from_id / to_id
    """
    key = standardize_id(shape.ID)
    txt = shape.text.strip() or "Donnée sans nom"
    data_type = "déclenchante" if layer == "T link" else "nourrissante"

    d = Data.query.filter_by(shape_id=key, type=data_type).first()
    if d:
        old = d.name
        if old != txt:
            d.name = txt
            rename_summaries.append((old, txt))
            db.session.commit()
            print(f"INFO : Connector rename: (ID={d.id}) '{old}' => '{txt}'")
    else:
        d = Data(name=txt, type=data_type, shape_id=key, layer=layer)
        db.session.add(d)
        db.session.flush()
        print(f"INFO : Connector créé => '{txt}' (ID={d.id}, shape_id={key})")

    data_mapping[key] = d.id
    db.session.commit()

    # On récupère from_id / to_id sans créer de link
    conns = analyze_connections(shape)
    from_id = conns.get("from_id")
    to_id = conns.get("to_id")

    connectors_list.append({
        "data_id": d.id,
        "data_name": d.name,
        "data_type": data_type,
        "from_raw": from_id,
        "to_raw": to_id
    })


def remove_data_not_in_new_mapping():
    existing_data = Data.query.filter(Data.shape_id.isnot(None)).all()
    count = 0
    for d in existing_data:
        sid = d.shape_id
        if sid not in data_mapping and sid not in return_mapping:
            print(f"INFO : Suppression data '{d.name}' (ID={d.id}, type={d.type},
shape_id={sid})")
            db.session.delete(d)
            count += 1
    db.session.commit()
    return count


##############################################################
```

```
################
# C) Reconstruction des liens
################################################################
################

def rebuild_links_from_connectors():
    """
    Pour chaque connecteur stocké dans connectors_list, on crée un unique Link
    (source=..., target=..., description=data_name) si from/to pointent vers
    des entités valides (Activity ou Data).
    """
    for c in connectors_list:
        data_id = c["data_id"]
        data_name = c["data_name"]
        data_type = c["data_type"]
        from_raw = c["from_raw"]
        to_raw = c["to_raw"]

        if not from_raw or not to_raw or (from_raw == to_raw):
            print(f"INFO : Connecteur partiel/boucle => '{data_name}', on ignore.")
            continue

        (skind, sid) = resolve_visio_id(from_raw)
        (tkind, tid) = resolve_visio_id(to_raw)

        if not sid or not tid or sid == tid:
            print(f"INFO : Connecteur impossible => '{data_name}' => on ignore")
            continue

        create_single_link(data_id, data_name, data_type, skind, sid, tkind, tid)


def create_single_link(data_id, data_name, data_type, skind, sid, tkind, tid):
    """
    Crée un Link (description=data_name) reliant
    source(activity/data) => target(activity/data),
    si le lien n'existe pas déjà.
    """
    s_name = get_entity_name(sid, skind)
    t_name = get_entity_name(tid, tkind)

    new_link = Link(type=data_type, description=data_name)
    if skind == 'activity':
        new_link.source_activity_id = sid
    else:
        new_link.source_data_id = sid

    if tkind == 'activity':
        new_link.target_activity_id = tid
    else:
        new_link.target_data_id = tid

    # Vérif duplication
    q = Link.query.filter_by(type=data_type, description=data_name)
    if new_link.source_activity_id:
        q = q.filter_by(source_activity_id=new_link.source_activity_id)
    else:
        q = q.filter_by(source_data_id=new_link.source_data_id)

    if new_link.target_activity_id:
        q = q.filter_by(target_activity_id=new_link.target_activity_id)
    else:
        q = q.filter_by(target_data_id=new_link.target_data_id)

    if q.first():
        print(f"INFO : Lien déjà existant => {s_name} -> {t_name} (data='{data_name}') => on
ignore")
        return

    db.session.add(new_link)
    db.session.flush()

    link_summaries.append((data_name, data_type, s_name, t_name))
    print(f"INFO : Lien créé => {s_name} -> {t_name} (data='{data_name}')")


##################################################################
################
# D) Nettoyage final + Récap
##################################################################
################

def cleanup_orphan_links():
    """
    En théorie, si on reconstruit tout, plus grand-chose orphelin.
    Mais on fait un check final si un lien pointe sur un ID inexistant.
    """
    all_links = Link.query.all()
    removed = 0
    for lk in all_links:
        remove_this = False
        if lk.source_activity_id and not Activities.query.get(lk.source_activity_id):
            remove_this = True
        if lk.source_data_id and not Data.query.get(lk.source_data_id):
            remove_this = True
        if lk.target_activity_id and not Activities.query.get(lk.target_activity_id):
            remove_this = True
        if lk.target_data_id and not Data.query.get(lk.target_data_id):
            remove_this = True
        if remove_this:
            print(f"INFO : Suppression lien orphelin ID={lk.id}, desc='{lk.description}'")
            db.session.delete(lk)
            removed += 1

    if removed > 0:
        db.session.commit()
        print(f"INFO : {removed} lien(s) orphelin(s) supprimé(s).")


def get_entity_name(eid, kind):
    """Renvoie le .name de l’activité ou du data pour logs."""
    if not eid:
        return "??"
    if kind == 'activity':
        a = Activities.query.get(eid)
        return a.name if a else "activité_inconnue"
    elif kind == 'data':
        dd = Data.query.get(eid)
        return dd.name if dd else "data_inconnue"
    return "inconnu"


def get_layer(shape):
    cell = shape.xml.find(".//{*}Cell[@N='LayerMember']")
    if cell is not None:
        val = cell.get("V")
        return LAYER_MAPPING.get(val, val)
    return None
```

```
def get_fill_pattern(shape):
    cell = shape.xml.find(".//{*}Cell[@N='FillPattern']")
    if cell is not None:
        return cell.get("V")
    return None


def analyze_connections(shape):
    """Retourne {'from_id':..., 'to_id':...}"""
    conns = {"from_id": None, "to_id": None}
    for cell in shape.xml.findall(".//{*}Cell"):
        n = cell.get("N")
        f = cell.get("F")
        if n == "BeginX":
            conns["from_id"] = extract_shape_id(f)
        elif n == "EndX":
            conns["to_id"] = extract_shape_id(f)
    return conns


def extract_shape_id(f_val):
    if f_val and "Sheet." in f_val:
        try:
            return int(f_val.split("Sheet.")[1].split("!")[0])
        except:
            return None
    return None


def resolve_visio_id(raw_id):
    """
    Convertit l'int 'raw_id' en (kind, db_id).
    S'il s'agit d'un Return, on tente d'associer l'activité correspondante si possible.
    """
    if not raw_id:
        return (None, None)
    key = str(raw_id).lower()  # plus simple qu'un standardize_id

    # 1) Activity
    if key in activity_mapping:
        return ('activity', activity_mapping[key])
    # 2) Data normal
    if key in data_mapping:
        return ('data', data_mapping[key])

    # 3) Data 'Return'
    if key in return_mapping:
        d_id = return_mapping[key]
        d = Data.query.get(d_id)
        if d and d.type.lower() == "retour":
            # Chercher activity portant le même name
            same_act = Activities.query.filter_by(name=d.name).first()
            if same_act:
                return ('activity', same_act.id)
        return ('data', d_id)

    return (None, None)


def standardize_id(visio_id):
    """Convertit shape.ID en string stable (p.ex "10")."""
    try:
        return str(int(visio_id)).strip().lower()
    except:
        return str(visio_id).strip().lower()


def print_summary():
    print("\n--- RÉSUMÉ DES LIENS CRÉÉS ---")
    if link_summaries:
        for (data_name, data_type, s_name, t_name) in link_summaries:
            print(f"  - '{data_name}' ({data_type}) : {s_name} -> {t_name}")
    else:
        print("  Aucun lien créé")
    print("--- Fin du résumé ---\n")

    if rename_summaries:
        print("--- Renommages détectés ---")
        for (old, new) in rename_summaries:
            print(f"  * '{old}' => '{new}'")
        print("--- Fin des renommages ---\n")

    print("CONFIRMATION : toutes les opérations ont été effectuées avec succès.")
```

```
def remove_activities_not_in_new_mapping():
    existing_acts = Activities.query.filter(Activities.shape_id.isnot(None)).all()
    count = 0
    for act in existing_acts:
        if act.shape_id not in activity_mapping:
            print(f"INFO : Suppression Activity '{act.name}' ID={act.id}, shape_id={act.shape_id}")
            db.session.delete(act)
            count += 1
    db.session.commit()
    return count


def remove_data_not_in_new_mapping():
    existing_data = Data.query.filter(Data.shape_id.isnot(None)).all()
    count = 0
    for d in existing_data:
        if d.shape_id not in data_mapping and d.shape_id not in return_mapping:
            print(f"INFO : Suppression Data '{d.name}' ID={d.id}, shape_id={d.shape_id}")
            db.session.delete(d)
            count += 1
    db.session.commit()
    return count


if __name__ == "__main__":
    from flask import Flask
    app = create_app()
    with app.app_context():
        vsdx_path = os.path.join("Code", "example.vsdx")  # adapter si besoin
        process_visio_file(vsdx_path)
```

__init__.py

env.py

```
import logging
from logging.config import fileConfig

from flask import current_app

from alembic import context
```

```
# this is the Alembic Config object, which provides
# access to the values within the .ini file in use.
config = context.config

# Interpret the config file for Python logging.
# This line sets up loggers basically.
fileConfig(config.config_file_name)
logger = logging.getLogger('alembic.env')


def get_engine():
    try:
        # this works with Flask-SQLAlchemy<3 and Alchemical
        return current_app.extensions['migrate'].db.get_engine()
    except (TypeError, AttributeError):
        # this works with Flask-SQLAlchemy>=3
        return current_app.extensions['migrate'].db.engine


def get_engine_url():
    try:
        return get_engine().url.render_as_string(hide_password=False).replace(
            '%', '%%')
    except AttributeError:
        return str(get_engine().url).replace('%', '%%')


# add your model's MetaData object here
# for 'autogenerate' support
# from myapp import mymodel
# target_metadata = mymodel.Base.metadata
config.set_main_option('sqlalchemy.url', get_engine_url())
target_db = current_app.extensions['migrate'].db

# other values from the config, defined by the needs of env.py,
# can be acquired:
# my_important_option = config.get_main_option("my_important_option")
# ... etc.


def get_metadata():
    if hasattr(target_db, 'metadatas'):
        return target_db.metadatas[None]
    return target_db.metadata


def run_migrations_offline():
    """Run migrations in 'offline' mode.

    This configures the context with just a URL
    and not an Engine, though an Engine is acceptable
    here as well.  By skipping the Engine creation
    we don't even need a DBAPI to be available.

    Calls to context.execute() here emit the given string to the
    script output.

    """
    url = config.get_main_option("sqlalchemy.url")
    context.configure(
        url=url, target_metadata=get_metadata(), literal_binds=True
    )

    with context.begin_transaction():
        context.run_migrations()


def run_migrations_online():
    """Run migrations in 'online' mode.

    In this scenario we need to create an Engine
    and associate a connection with the context.

    """

    # this callback is used to prevent an auto-migration from being generated
    # when there are no changes to the schema
    # reference: http://alembic.zzzcomputing.com/en/latest/cookbook.html
    def process_revision_directives(context, revision, directives):
        if getattr(config.cmd_opts, 'autogenerate', False):
            script = directives[0]
            if script.upgrade_ops.is_empty():
                directives[:] = []
                logger.info('No changes in schema detected.')
    conf_args = current_app.extensions['migrate'].configure_args
    if conf_args.get("process_revision_directives") is None:
        conf_args["process_revision_directives"] = process_revision_directives

    connectable = get_engine()

    with connectable.connect() as connection:
        context.configure(
            connection=connection,
            target_metadata=get_metadata(),
            **conf_args
        )

        with context.begin_transaction():
            context.run_migrations()


if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

37f4fdd5526b_migration_initiale_avec_la_nouvelle_.py

```
"""Migration initiale avec la nouvelle table Link

Revision ID: 37f4fdd5526b
Revises:
Create Date: 2025-02-25 19:45:51.119227

"""
from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision = '37f4fdd5526b'
down_revision = None
branch_labels = None
depends_on = None


def upgrade():
    # ### commands auto generated by Alembic - please adjust! ###
```

```
op.create_table('activities',
sa.Column('id', sa.Integer(), nullable=False),
sa.Column('shape_id', sa.String(length=50), nullable=True),
sa.Column('name', sa.String(length=200), nullable=False),
sa.Column('description', sa.Text(), nullable=True),
sa.Column('is_result', sa.Boolean(), nullable=False),
sa.PrimaryKeyConstraint('id')
)
with op.batch_alter_table('activities', schema=None) as batch_op:
    batch_op.create_index(batch_op.f('ix_activities_shape_id'), ['shape_id'], unique=True)

op.create_table('data',
sa.Column('id', sa.Integer(), nullable=False),
sa.Column('shape_id', sa.String(length=50), nullable=True),
sa.Column('name', sa.String(length=255), nullable=False),
sa.Column('type', sa.String(length=50), nullable=False),
sa.Column('description', sa.Text(), nullable=True),
sa.Column('layer', sa.String(length=50), nullable=True),
sa.PrimaryKeyConstraint('id')
)
with op.batch_alter_table('data', schema=None) as batch_op:
    batch_op.create_index(batch_op.f('ix_data_shape_id'), ['shape_id'], unique=True)

op.create_table('roles',
sa.Column('id', sa.Integer(), nullable=False),
sa.Column('name', sa.String(length=100), nullable=False),
sa.PrimaryKeyConstraint('id'),
sa.UniqueConstraint('name')
)
op.create_table('tools',
sa.Column('id', sa.Integer(), nullable=False),
sa.Column('name', sa.String(length=255), nullable=False),
sa.Column('description', sa.Text(), nullable=True),
sa.PrimaryKeyConstraint('id'),
sa.UniqueConstraint('name')
)
op.create_table('activity_roles',
sa.Column('activity_id', sa.Integer(), nullable=False),
sa.Column('role_id', sa.Integer(), nullable=False),
sa.Column('status', sa.String(length=50), nullable=False),
sa.ForeignKeyConstraint(['activity_id'], ['activities.id'], ),
sa.ForeignKeyConstraint(['role_id'], ['roles.id'], ),
sa.PrimaryKeyConstraint('activity_id', 'role_id')
```

```
)
op.create_table('competencies',
sa.Column('id', sa.Integer(), nullable=False),
sa.Column('description', sa.Text(), nullable=False),
sa.Column('activity_id', sa.Integer(), nullable=False),
sa.ForeignKeyConstraint(['activity_id'], ['activities.id'], ),
sa.PrimaryKeyConstraint('id')
)
op.create_table('links',
sa.Column('id', sa.Integer(), nullable=False),
sa.Column('source_activity_id', sa.Integer(), nullable=True),
sa.Column('source_data_id', sa.Integer(), nullable=True),
sa.Column('target_activity_id', sa.Integer(), nullable=True),
sa.Column('target_data_id', sa.Integer(), nullable=True),
sa.Column('type', sa.String(length=50), nullable=False),
sa.Column('description', sa.Text(), nullable=True),
sa.ForeignKeyConstraint(['source_activity_id'], ['activities.id'], ),
sa.ForeignKeyConstraint(['source_data_id'], ['data.id'], ),
sa.ForeignKeyConstraint(['target_activity_id'], ['activities.id'], ),
sa.ForeignKeyConstraint(['target_data_id'], ['data.id'], ),
sa.PrimaryKeyConstraint('id')
)
op.create_table('softskills',
sa.Column('id', sa.Integer(), nullable=False),
sa.Column('habilete', sa.String(length=255), nullable=False),
sa.Column('niveau', sa.String(length=10), nullable=False),
sa.Column('activity_id', sa.Integer(), nullable=False),
sa.ForeignKeyConstraint(['activity_id'], ['activities.id'], ),
sa.PrimaryKeyConstraint('id')
)
op.create_table('tasks',
sa.Column('id', sa.Integer(), nullable=False),
sa.Column('name', sa.String(length=255), nullable=False),
sa.Column('description', sa.Text(), nullable=True),
sa.Column('order', sa.Integer(), nullable=True),
sa.Column('activity_id', sa.Integer(), nullable=False),
sa.ForeignKeyConstraint(['activity_id'], ['activities.id'], ),
sa.PrimaryKeyConstraint('id')
)
op.create_table('task_roles',
sa.Column('task_id', sa.Integer(), nullable=False),
sa.Column('role_id', sa.Integer(), nullable=False),
sa.Column('status', sa.String(length=50), nullable=False),
    sa.ForeignKeyConstraint(['role_id'], ['roles.id'], ),
    sa.ForeignKeyConstraint(['task_id'], ['tasks.id'], ),
    sa.PrimaryKeyConstraint('task_id', 'role_id')
    )
    op.create_table('task_tools',
    sa.Column('task_id', sa.Integer(), nullable=False),
    sa.Column('tool_id', sa.Integer(), nullable=False),
    sa.ForeignKeyConstraint(['task_id'], ['tasks.id'], ),
    sa.ForeignKeyConstraint(['tool_id'], ['tools.id'], ),
    sa.PrimaryKeyConstraint('task_id', 'tool_id')
    )
    # ### end Alembic commands ###


def downgrade():
    # ### commands auto generated by Alembic - please adjust! ###
    op.drop_table('task_tools')
    op.drop_table('task_roles')
    op.drop_table('tasks')
    op.drop_table('softskills')
    op.drop_table('links')
    op.drop_table('competencies')
    op.drop_table('activity_roles')
    op.drop_table('tools')
    op.drop_table('roles')
    with op.batch_alter_table('data', schema=None) as batch_op:
        batch_op.drop_index(batch_op.f('ix_data_shape_id'))

    op.drop_table('data')
    with op.batch_alter_table('activities', schema=None) as batch_op:
        batch_op.drop_index(batch_op.f('ix_activities_shape_id'))

    op.drop_table('activities')
    # ### end Alembic commands ###
```

optiq.css

```
/* ====== Reprise complète du style de la dernière version fonctionnelle ====== */

/* Corps de page */
body {
  font-family: Arial, sans-serif;
}
```

```
/* Bouton de mise à jour de la cartographie */
.update-btn {
  background-color: green;
  color: white;
  padding: 10px 15px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  margin-bottom: 20px;
  font-size: 1em;
}

/* Conteneur principal de chaque activité */
.activity-container {
  border: 1px solid #aaa;
  margin-bottom: 20px;
  overflow: hidden;
}

/* En-tête de l'activité, police réduite */
.activity-header {
  background-color: #add8e6;
  padding: 5px;
  font-size: 0.8em; /* police un peu réduite */
  cursor: pointer;
  display: flex;
  align-items: center;
}
.activity-header h2 {
  margin: 0;
  flex-grow: 1;
  font-size: 1em;
}
.toggle-icon {
  font-size: 1em;
  margin-right: 5px;
}

/* Détails de l'activité masqués par défaut */
.activity-details {
  padding: 5px;
  display: none;
}
```

```
/* Connexions entrantes/sortantes : conteneur + titres */
.connections-container {
  display: flex;
  justify-content: space-between;
  flex-wrap: wrap;
  margin-top: 10px;
}
.connections-container > div {
  width: 48%;
}
/* Titre plus petit pour "Connexions entrantes" et "Connexions sortantes" */
.connections-container h3 {
  font-size: 0.9em;
  margin: 0 0 5px 0;
}

/* Tableau de connexions */
.conn-table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: 10px;
}
.conn-table th,
.conn-table td {
  border: 1px solid #ccc;
  padding: 4px;
  text-align: left;
}
.conn-table th {
  background-color: #cce5ff;
}

/* Mise en forme conditionnelle : déclenchante/nourrissante */
.declenchante {
  font-weight: bold;
}
.nourrissante {
  font-style: italic;
}

/* Section Tâches */
.tasks-section {
  margin-top: 10px;
}
.tasks-section h3 {
  font-size: 0.9em;
  margin-bottom: 5px;
}

/* Conteneur de chaque tâche */
.task {
  border: 1px solid #ddd;
  padding: 5px;
  margin-bottom: 10px;
  display: flex;
  flex-direction: column;
}

/* Ligne principale de la tâche */
.task-row {
  display: flex;
  justify-content: space-between;
  align-items: flex-start;
}
.task-left {
  display: flex;
  align-items: center;
  gap: 5px;
  flex: 1;
}
.task-right {
  display: flex;
  flex-direction: column;
  align-items: flex-start;
  gap: 5px;
  max-width: 250px;
}

/* Titre de la tâche (nom + description) */
.task-title {
  max-width: 45ch;
  word-wrap: break-word;
}

/* Liste des outils : inline-block pour les aligner sur une ligne, text-align: left */
```

```
.tools-list ul {
  list-style: none;
  padding: 0;
  margin: 0;
  text-align: left;
}
.tools-list li {
  background: #f0f0f0;
  padding: 2px 5px;
  border-radius: 3px;
  margin-bottom: 5px;
  display: inline-block; /* pour aligner sur une ligne */
  margin-right: 5px;
}

/* Bouton d'ajout d'outil : stylé plus petit, mieux aligné */
.add-tool-btn {
  width: 24px;
  height: 24px;
  border: 1px solid #ccc;
  background-color: #fafafa;
  display: inline-flex;
  align-items: center;
  justify-content: center;
  vertical-align: middle;
  margin: 0;
  padding: 0;
}

/* Formulaire d'édition de tâche */
.edit-task-form {
  display: none;
  margin-top: 5px;
  border: 1px solid #ccc;
  padding: 5px;
  width: 100%;
}

/* Formulaires d'ajout de tâche / d'outils */
.task-form,
.tool-form {
  margin-top: 5px;
  border: 1px solid #ccc;
  padding: 5px;
}

/* Boutons icônes (poubelle, crayon, etc.) */
.icon-btn {
  background: none;
  border: none;
  cursor: pointer;
  margin-left: 5px;
}
.icon-btn i {
  font-size: 1em;
}
```

competencies.js

```
// static/js/competencies.js

/**
 * Ajoute une compétence dans le DOM (après /skills/add).
 */
function addCompetencyItemToDOM(activityId, compId, description) {
    const container = document.getElementById(`competencies-list-${activityId}`);
    if (!container) return;
    const li = document.createElement('li');
    li.setAttribute('data-comp-id', compId);
    li.style.marginBottom = "5px";
    li.innerHTML = `
      <span class="validated-skill-text">${description}</span>
      <button onclick="editCompetency(this, ${compId})" style="margin-left:5px;">
        <i class="fa-solid fa-pencil"></i>
      </button>
      <button onclick="deleteCompetency(this, ${compId})" style="margin-left:5px;">
        <i class="fa-solid fa-trash"></i>
      </button>
      <div class="edit-competency-form" id="edit-competency-form-${compId}"
style="display:none; margin-top:5px;">
        <label>Description :</label>
        <input type="text" id="edit-competency-desc-${compId}" value="${description}" />
        <button onclick="submitEditCompetency('${compId}')">Enregistrer</button>
        <button onclick="hideEditCompetencyForm('${compId}')">Annuler</button>
      </div>
    `;
    container.appendChild(li);
 }

 // Ouvre le formulaire d'édition
 function editCompetency(buttonElem, compId) {
   document.getElementById(`edit-competency-form-${compId}`).style.display = "block";
 }

 // Ferme le formulaire d'édition
 function hideEditCompetencyForm(compId) {
   document.getElementById(`edit-competency-form-${compId}`).style.display = "none";
 }

 // Soumet la mise à jour (PUT /skills/<compId>)
 function submitEditCompetency(compId) {
   const newDesc = document.getElementById(`edit-competency-desc-
${compId}`).value.trim();
   if (!newDesc) {
     alert("Veuillez saisir une description.");
     return;
   }
   fetch(`/skills/${compId}`, {
     method: 'PUT',
     headers: { 'Content-Type': 'application/json' },
     body: JSON.stringify({ description: newDesc })
   })
   .then(res => res.json())
   .then(data => {
     if (data.error) {
       alert("Erreur : " + data.error);
     } else {
       // On met à jour le texte dans le DOM
       const li = document.querySelector(`li[data-comp-id='${compId}']`);
       if (li) {
         li.querySelector('.validated-skill-text').textContent = data.description;
         hideEditCompetencyForm(compId);
         alert("Compétence mise à jour en base.");
       }
     }
   })
   .catch(err => {
     alert("Erreur lors de la mise à jour : " + err.message);
   });
 }
 // Supprime une compétence (DELETE /skills/<compId>)
 function deleteCompetency(buttonElem, compId) {
  if (!confirm("Confirmez-vous la suppression de cette compétence ?")) return;
  fetch(`/skills/${compId}`, { method: 'DELETE' })
  .then(res => res.json())
  .then(data => {
   if (data.error) {
    alert("Erreur : " + data.error);
   } else {
    const li = buttonElem.parentNode;
    li.parentNode.removeChild(li);
    alert("Compétence supprimée.");
   }
  })
  .catch(err => {
   alert("Erreur lors de la suppression : " + err.message);
  });
 }
```

main.js

```
// main.js - Fonctions globales

// Fonction pour ouvrir/fermer les activités
function toggleDetails(detailsId, headerElem) {
  console.log("toggleDetails called with detailsId =", detailsId);
  const detailsElem = document.getElementById(detailsId);
  const iconElem = headerElem.querySelector('.toggle-icon');
  const currentDisplay = window.getComputedStyle(detailsElem).display;
  if (currentDisplay === "none") {
   detailsElem.style.display = "block";
   iconElem.textContent = "▼";
  } else {
   detailsElem.style.display = "none";
   iconElem.textContent = "▶";
  }
 }
 console.log("toggleDetails loaded in main.js");

 // Fonctions globales pour récupérer les informations d'une activité
 function getActivityDetails(activityId) {
   const detailsElem = document.getElementById('details-' + activityId);
   return detailsElem ? detailsElem.innerText : "";
 }

 function getCompetenciesData(activityId) {
    const compElem = document.getElementById('competencies-' + activityId);
    return compElem ? compElem.innerText : "";
 }
```

roles.js

```
// static/js/roles.js

function openGarantModal(activityId) {
   document.getElementById('garantModal').style.display = 'block';
   document.getElementById('garant-activity-id').value = activityId;

   // Charger la liste des rôles
   let selectElem = document.getElementById('garant-role-select');
   selectElem.innerHTML = "";

   fetch('/roles/list')
   .then(response => response.json())
   .then(data => {
      data.forEach(r => {
         let opt = document.createElement('option');
         opt.value = r.name;
         opt.textContent = r.name;
         selectElem.appendChild(opt);
      });
   })
   .catch(err => {
      alert("Erreur lors du chargement des rôles: " + err.message);
   });
}

function closeGarantModal() {
   document.getElementById('garantModal').style.display = 'none';
}

function submitGarantRole() {
   let activityId = document.getElementById('garant-activity-id').value;
   let selectElem = document.getElementById('garant-role-select');
   let newRoleInput = document.getElementById('garant-new-role').value.trim();
   let roleName = newRoleInput || selectElem.value;
```

```
  if(!roleName) {
    alert("Veuillez sélectionner ou saisir un rôle.");
    return;
  }

  fetch('/roles/garant/activity/' + activityId, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ role_name: roleName })
  })
  .then(r => r.json())
  .then(data => {
    if(data.error) {
      alert("Erreur: " + data.error);
    } else {
      // Mettre à jour l'affichage
      let garantSpan = document.getElementById('activity-garant-' + activityId);
      garantSpan.textContent = "Garant : " + data.role.name;
      closeGarantModal();
    }
  })
  .catch(err => {
    alert("Erreur lors de l'enregistrement du rôle: " + err.message);
  });
}
```

softskills.js

```
// static/js/softskills.js

function getActivityDetails(activityId) {
  const detailsElem = document.getElementById('details-' + activityId);
  return detailsElem ? detailsElem.innerText : "";
 }

 function getCompetenciesData(activityId) {
  const compElem = document.getElementById('competencies-list-' + activityId);
  if (!compElem) return "";
  let items = [];
  compElem.querySelectorAll('li').forEach(li => items.push(li.textContent.trim()));
  return items.join(", ");
 }

 function translateLevelToText(level) {
  switch(level) {
    case "1": return "Aptitude";
    case "2": return "Acquisition";
    case "3": return "Maîtrise";
    case "4": return "Excellence";
    default:  return "Inconnu";
  }
}

// Bouton "Proposer HSC"
$(document).on('click', '.define-hsc-btn', function() {
  const activityId = $(this).data('activity-id');
  console.log("Proposer HSC pour activityId =", activityId);
  proposeSoftskills(activityId);
});

function proposeSoftskills(activityId) {
  const activityData = getActivityDetails(activityId);
  const competenciesData = getCompetenciesData(activityId);
  fetch('/softskills/propose', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      activity: activityData,
      competencies: competenciesData
    })
  })
  .then(res => res.json())
  .then(data => {
    console.log("Réponse /softskills/propose:", data);
    if (data.error) {
      alert(data.error);
      return;
    }
    let addPromises = [];
    data.forEach(item => {
      let p = fetch('/softskills/add', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          activity_id: activityId,
          habilete: item.habilete,
          niveau: item.niveau
        })
      })
      .then(r => r.json())
      .then(added => {
        if (!added.error) {
          addSoftskillItemToDOM(activityId, added.habilete, added.niveau, added.id);
        } else {
          console.error("Erreur ajout HSC:", added.error);
        }
      })
      .catch(err => console.error("Erreur fetch /softskills/add:", err));
      addPromises.push(p);
    });
    return Promise.all(addPromises);
  })
  .catch(err => {
    alert("Erreur lors de la proposition HSC : " + err.message);
    console.error(err);
  });
}

// AJOUT MANUEL
function submitSoftskill(activityId) {
  console.log("submitSoftskill pour activityId =", activityId);
  const nameInput = document.getElementById('softskill-name-' + activityId);
  const levelInput = document.getElementById('softskill-level-' + activityId);
  const hscName = nameInput.value.trim();
  const hscLevel = levelInput.value.trim();
  if(!hscName) {
    alert("Veuillez saisir un nom d'habileté.");
    return;
  }
  if(!["1","2","3","4"].includes(hscLevel)) {
    alert("Le niveau doit être 1, 2, 3 ou 4.");
    return;
  }
  fetch('/softskills/add', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      activity_id: activityId,
      habilete: hscName,
      niveau: hscLevel
```

```
    })
  })
  .then(res => res.json())
  .then(data => {
    if (data.error) {
      alert("Erreur : " + data.error);
    } else {
      addSoftskillItemToDOM(activityId, data.habilete, data.niveau, data.id);
      nameInput.value = "";
      levelInput.value = "";
      alert("Habileté sauvegardée en base.");
    }
  })
  .catch(err => {
    alert("Erreur lors de l'ajout : " + err.message);
    console.error(err);
  });
}

// Ajoute une HSC dans le DOM
function addSoftskillItemToDOM(activityId, hscName, hscLevel, dbId) {
  console.log("Ajout DOM HSC:", { activityId, hscName, hscLevel, dbId });
  const container = document.getElementById('softskills-list-' + activityId);
  if (!container) {
    console.error("Conteneur #softskills-list-" + activityId + " introuvable.");
    return;
  }
  const levelLabel = translateLevelToText(hscLevel);
  const div = document.createElement('div');
  div.className = 'softskill-item';
  div.style.marginBottom = '5px';
  div.setAttribute('data-ss-id', dbId);
  div.innerHTML = `
    <span class="softskill-text">${hscName} (Niveau: <span class="softskill-
level">${levelLabel}</span>)</span>
    <i class="fas fa-pencil-alt edit-softskill" title="Modifier"></i>
    <i class="fas fa-trash delete-softskill" title="Supprimer"></i>
    <div class="edit-softskill-form" id="edit-softskill-form-${dbId}" style="display:none;">
      <label>Habileté :</label>
      <input type="text" id="edit-softskill-name-${dbId}" value="${hscName}" />
      <label>Niveau (1..4) :</label>
      <input type="number" min="1" max="4" id="edit-softskill-level-${dbId}"
value="${hscLevel}" />
```

```
    <button onclick="submitEditSoftskillFromDOM('${dbId}')">Enregistrer</button>
    <button onclick="hideEditSoftskillForm('${dbId}')">Annuler</button>
   </div>
  `;
  container.appendChild(div);
}

// Edition
$(document).on('click', '.edit-softskill', function() {
  const itemElem = $(this).closest('.softskill-item');
  const dbId = itemElem.data('ss-id');
  console.log("Edition HSC dbId =", dbId);
  document.getElementById(`edit-softskill-form-${dbId}`).style.display = 'block';
});

function hideEditSoftskillForm(dbId) {
  document.getElementById(`edit-softskill-form-${dbId}`).style.display = 'none';
}

function submitEditSoftskillFromDOM(dbId) {
  console.log("submitEditSoftskillFromDOM dbId =", dbId);
  const newName = document.getElementById(`edit-softskill-name-${dbId}`).value.trim();
  const newLevel = document.getElementById(`edit-softskill-level-${dbId}`).value.trim();
  if(!newName) {
    alert("Veuillez saisir un nom d'habileté.");
    return;
  }
  if(!["1","2","3","4"].includes(newLevel)) {
    alert("Le niveau doit être 1, 2, 3 ou 4.");
    return;
  }
  fetch(`/softskills/${dbId}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ habilete: newName, niveau: newLevel })
  })
  .then(res => res.json())
  .then(data => {
    console.log("Réponse PUT /softskills:", data);
    if (data.error) {
      alert("Erreur : " + data.error);
    } else {
      const itemElem = document.querySelector(`.softskill-item[data-ss-id='${dbId}']`);
      if (itemElem) {
        const textElem = itemElem.querySelector('.softskill-text');
        const levelLabel = translateLevelToText(data.niveau);
        textElem.innerHTML = `${data.habilete} (Niveau: <span class="softskill-
level">${levelLabel}</span>)`;
        hideEditSoftskillForm(dbId);
        alert("HSC mise à jour en base.");
      }
    }
  })
  .catch(err => {
    alert("Erreur lors de la mise à jour : " + err.message);
    console.error(err);
  });
}

// Suppression
$(document).on('click', '.delete-softskill', function() {
  const itemElem = $(this).closest('.softskill-item');
  const dbId = itemElem.data('ss-id');
  console.log("deleteSoftskill dbId =", dbId);
  if(!confirm("Voulez-vous supprimer cette habileté ?")) return;
  fetch(`/softskills/${dbId}`, { method: 'DELETE' })
  .then(res => res.json())
  .then(data => {
    if (data.error) {
      alert("Erreur : " + data.error);
    } else {
      itemElem.remove();
      alert("HSC supprimée.");
    }
  })
  .catch(err => {
    alert("Erreur lors de la suppression : " + err.message);
    console.error(err);
  });
});
```

tasks.js

```
/* tasks.js - Gestion des tâches */

// Affiche le formulaire d'ajout d'une tâche pour une activité donnée
function showTaskForm(activityId) {
  document.getElementById('task-form-' + activityId).style.display = 'block';
}

// Cache le formulaire d'ajout d'une tâche pour une activité donnée
function hideTaskForm(activityId) {
  document.getElementById('task-form-' + activityId).style.display = 'none';
}

// Soumet une nouvelle tâche pour une activité
function submitTask(activityId) {
  const taskName = document.getElementById('task-name-' + activityId).value;
  const taskDesc = document.getElementById('task-desc-' + activityId).value;
  if (!taskName) {
    alert("Le nom de la tâche est requis.");
    return;
  }
  fetch('/activities/' + activityId + '/tasks/add', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: taskName, description: taskDesc })
  })
  .then(response => {
    if (response.ok) return response.json();
    throw new Error("Erreur lors de l'ajout de la tâche.");
  })
  .then(data => {
    let tasksList = document.getElementById('tasks-list-' + activityId);
    if (!tasksList) {
      tasksList = document.createElement('ul');
      tasksList.id = 'tasks-list-' + activityId;
      const tasksSection = document.getElementById('task-form-' + activityId).parentNode;
      tasksSection.insertBefore(tasksList, tasksSection.firstChild);
    }
    const li = document.createElement('li');
    li.className = 'task';
    li.id = 'task-' + data.id;
    li.setAttribute('data-task-id', data.id);
    li.innerHTML = `
      <div class="task-row">
        <div class="task-left">
          <i class="fa-solid fa-bars icon-btn" style="cursor: move;"></i>
          <span class="task-title">
            <strong id="task-name-display-${data.id}">${data.name}</strong>
      ${data.description ? ' - <span id="task-desc-display-' + data.id + '">' + data.description
+ '</span>' : ''}
     </span>
     <button class="icon-btn" onclick="deleteTask('${activityId}', '${data.id}')">
      <i class="fa-solid fa-trash"></i>
     </button>
     <button class="icon-btn" onclick="showEditTaskForm('${activityId}', '${data.id}',
'${data.name}', '${data.description || ''}')">
      <i class="fa-solid fa-pencil"></i>
     </button>
    </div>
    <div class="task-right">
     <div class="tools-list" id="tools-for-task-${data.id}">
      <ul>
       <li id="no-tools-msg-${data.id}">Aucun outil associé.</li>
       <li class="add-tool-li">
        <button class="icon-btn add-tool-btn" onclick="showToolForm('${data.id}')">
         <i class="fa-solid fa-plus"></i>
        </button>
       </li>
      </ul>
     </div>
    </div>
   </div>
   <div class="edit-task-form" id="edit-task-form-${data.id}">
    <input type="text" id="edit-task-name-${data.id}" placeholder="Nom de la tâche" />
    <input type="text" id="edit-task-desc-${data.id}" placeholder="Description
(optionnelle)" />
    <button onclick="submitEditTask('${activityId}', '${data.id}')">Enregistrer</button>
    <button onclick="hideEditTaskForm('${data.id}')">Annuler</button>
   </div>
   <div id="tool-form-${data.id}" class="tool-form" style="display: none;">
    <div>
     <label for="existing-tools-${data.id}">Outils existants:</label>
     <select id="existing-tools-${data.id}" multiple style="width: 100%;"></select>
    </div>
    <div>
     <label for="new-tools-${data.id}">Nouveaux outils (séparés par des virgules):</label>
     <input type="text" id="new-tools-${data.id}" placeholder="Ex: Outil1, Outil2"
style="width: 100%;" />
    </div>
    <button onclick="submitTools('${data.id}')">Enregistrer</button>
    <button onclick="hideToolForm('${data.id}')">Annuler</button>
    </div>
   `;
   tasksList.appendChild(li);
   document.getElementById('task-name-' + activityId).value = "";
   document.getElementById('task-desc-' + activityId).value = "";
   hideTaskForm(activityId);
  })
  .catch(error => {
   alert(error.message);
  });
}

// Supprime une tâche donnée
function deleteTask(activityId, taskId) {
  if (!confirm("Confirmez-vous la suppression de cette tâche ?")) return;
  fetch(`/activities/${activityId}/tasks/${taskId}`, { method: 'DELETE' })
  .then(response => response.json())
  .then(data => {
   const taskElem = document.getElementById('task-' + taskId);
   if (taskElem) {
    taskElem.parentNode.removeChild(taskElem);
   }
  })
  .catch(error => {
   alert(error.message);
  });
}

// Affiche le formulaire d'édition d'une tâche
function showEditTaskForm(activityId, taskId, name, description) {
  document.getElementById('edit-task-form-' + taskId).style.display = 'block';
  document.getElementById('edit-task-name-' + taskId).value = name;
  document.getElementById('edit-task-desc-' + taskId).value = description;
}

// Cache le formulaire d'édition d'une tâche
function hideEditTaskForm(taskId) {
  document.getElementById('edit-task-form-' + taskId).style.display = 'none';
}

// Soumet les modifications d'une tâche
function submitEditTask(activityId, taskId) {
  const newName = document.getElementById('edit-task-name-' + taskId).value;
  const newDesc = document.getElementById('edit-task-desc-' + taskId).value;
  if (!newName) {
    alert("Le nom de la tâche est requis.");
    return;
  }
  fetch(`/activities/${activityId}/tasks/${taskId}`, {
    method: 'PUT',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ name: newName, description: newDesc })
  })
  .then(response => response.json())
  .then(data => {
    const nameElem = document.getElementById('task-name-display-' + taskId);
    if (nameElem) {
      nameElem.textContent = data.name;
    }
    const descElem = document.getElementById('task-desc-display-' + taskId);
    if (descElem) {
      descElem.textContent = data.description;
    } else if (data.description) {
      const taskTitle = document.querySelector('#task-name-display-' + taskId).parentNode;
      const span = document.createElement('span');
      span.id = 'task-desc-display-' + taskId;
      span.textContent = data.description;
      taskTitle.appendChild(document.createTextNode(" - "));
      taskTitle.appendChild(span);
    }
    hideEditTaskForm(taskId);
  })
  .catch(error => {
    alert(error.message);
  });
}

/* --- DRAG & DROP POUR LES TÂCHES --- */
document.addEventListener('DOMContentLoaded', function() {
  const taskLists = document.querySelectorAll('[id^="tasks-list-"]');
  taskLists.forEach(list => {
    Sortable.create(list, {
      animation: 150,
      handle: '.fa-bars',
      onEnd: function(evt) {
        const listId = list.getAttribute('id'); // ex: tasks-list-123
        const activityId = listId.split('-')[2];   // ex: 123
        console.log("Reorder tasks for activityId=", activityId);
        let newOrder = [];
        list.querySelectorAll('li.task').forEach(taskElem => {
          newOrder.push(taskElem.getAttribute('data-task-id'));
        });
        fetch('/activities/' + activityId + '/tasks/reorder', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({ order: newOrder })
        }).then(function(response) {
          if (!response.ok) {
            console.error("Erreur de sauvegarde de l'ordre");
          }
        });
      }
    });
  });
});
```

tools.js

```
// tools.js - Gestion des outils

function showToolForm(taskId) {
    document.getElementById('tool-form-' + taskId).style.display = 'block';
    fetch('/tools/all')
    .then(response => response.json())
    .then(data => {
      const selectElem = document.getElementById('existing-tools-' + taskId);
      selectElem.innerHTML = "";
      data.forEach(tool => {
        const option = document.createElement('option');
        option.value = tool.id;
        option.text = tool.name;
        selectElem.appendChild(option);
      });
    })
    .catch(error => {
      alert("Erreur lors du chargement des outils existants: " + error.message);
    });
  }

  function hideToolForm(taskId) {
    document.getElementById('tool-form-' + taskId).style.display = 'none';
  }

  function submitTools(taskId) {
    const selectElem = document.getElementById('existing-tools-' + taskId);
    const newToolsInput = document.getElementById('new-tools-' + taskId);
    const existingToolIds = Array.from(selectElem.selectedOptions).map(opt =>
parseInt(opt.value));
    const newTools = newToolsInput.value.split(",").map(item => item.trim()).filter(item =>
item.length > 0);
    const payload = {
      task_id: parseInt(taskId),
      existing_tool_ids: existingToolIds,
      new_tools: newTools
    };
    fetch('/tools/add', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(payload)
    })
    .then(response => response.json())
    .then(data => {
      const noToolsMsg = document.getElementById('no-tools-msg-' + taskId);
      if (noToolsMsg) {
        noToolsMsg.parentNode.removeChild(noToolsMsg);
      }
      const toolsContainer = document.getElementById('tools-for-task-' + taskId);
      let ul = toolsContainer.querySelector('ul');
      if (!ul) {
        ul = document.createElement('ul');
        toolsContainer.appendChild(ul);
      }
      let addBtn = ul.querySelector('li.add-tool-li');
      if (addBtn) {
        addBtn.parentNode.removeChild(addBtn);
      }
      data.added_tools.forEach(tool => {
        const li = document.createElement('li');
        li.setAttribute("data-tool-id", tool.id);
        li.innerHTML = tool.name + ` <button class="icon-btn"
onclick="deleteToolFromTask('${taskId}', '${tool.id}')">
                                    <i class="fa-solid fa-trash"></i></button>`;
        ul.appendChild(li);
      });
      const newAddLi = document.createElement('li');
      newAddLi.className = 'add-tool-li';
      newAddLi.innerHTML = `<button class="icon-btn add-tool-btn"
onclick="showToolForm('${taskId}')">
                        <i class="fa-solid fa-plus"></i>
                      </button>`;
      ul.appendChild(newAddLi);
      selectElem.selectedIndex = -1;
      newToolsInput.value = "";
      hideToolForm(taskId);
    })
    .catch(error => {
      alert(error.message);
    });
  }

  function deleteToolFromTask(taskId, toolId) {
    if (!confirm("Confirmez-vous la suppression de cet outil ?")) return;
    fetch(`/activities/tasks/${taskId}/tools/${toolId}`, { method: 'DELETE' })
    .then(response => response.json())
    .then(data => {
      const toolsContainer = document.getElementById('tools-for-task-' + taskId);
      const ul = toolsContainer.querySelector('ul');
      if (!ul) return;
      const realToolElem = ul.querySelector(`li[data-tool-id="${toolId}"]`);
      if (realToolElem) {
        realToolElem.parentNode.removeChild(realToolElem);
      }
      let addBtn = ul.querySelector('li.add-tool-li');
      if (!addBtn) {
        const newAddLi = document.createElement('li');
        newAddLi.className = 'add-tool-li';
        newAddLi.innerHTML = `<button class="icon-btn add-tool-btn"
onclick="showToolForm('${taskId}')">
                          <i class="fa-solid fa-plus"></i>
                        </button>`;
        ul.appendChild(newAddLi);
      }
    })
    .catch(error => {
      alert(error.message);
    });
```

```
 }
```

translate_softskills.js

```
// translate_softskills.js - Gestion de la traduction des softskills en HSC

function openTranslateSoftskillsModal(activityId) {
  window.translateSoftskillsActivityId = activityId;
  document.getElementById('translateSoftskillsModal').style.display = 'block';
 }

 function closeTranslateSoftskillsModal() {
  document.getElementById('translateSoftskillsModal').style.display = 'none';
  window.translateSoftskillsActivityId = null;
 }

 // Soumet le texte entré pour traduction et ajoute les HSC traduites en base
 function submitSoftskillsTranslation() {
  const activityId = window.translateSoftskillsActivityId;
  if (!activityId) {
   alert("Identifiant de l'activité introuvable.");
   return;
  }
  const userInputElem = document.getElementById('translateSoftskillsInput');
  const userInput = userInputElem.value.trim();
  if (!userInput) {
   alert("Veuillez saisir du texte.");
   return;
  }

  // 1) Appel /softskills/translate pour obtenir un objet { "proposals": [...] }
  $.ajax({
   url: '/softskills/translate',
   method: 'POST',
   contentType: 'application/json',
   data: JSON.stringify({ user_input: userInput }),
   success: function(response) {
    // 'response' est un objet : { "proposals": [ { "habilete":"...", "niveau":"..." }, ... ] }
    if (!response.proposals) {
     alert("Réponse inattendue : pas de 'proposals' ?");
     return;
    }
    let addPromises = [];
    // 2) Pour chaque proposition, on fait un POST /softskills/add
```

```
      response.proposals.forEach(function(item) {
        let p = $.ajax({
          url: '/softskills/add',
          method: 'POST',
          contentType: 'application/json',
          data: JSON.stringify({
            activity_id: activityId,
            habilete: item.habilete,
            niveau: item.niveau
          }),
          success: function(added) {
            // 3) On insère la HSC dans le DOM
            if (added.error) {
              console.error("Erreur ajout HSC:", added.error);
            } else {
              addSoftskillItemToDOM(activityId, added.habilete, added.niveau, added.id);
            }
          },
          error: function(err) {
            console.error("Erreur /softskills/add:", err);
          }
        });
        addPromises.push(p);
      });

      // Quand toutes les requêtes sont terminées
      $.when.apply($, addPromises).then(function() {
        userInputElem.value = "";
        closeTranslateSoftskillsModal();
      });
    },
    error: function() {
      alert("Erreur lors de la traduction des softskills.");
    }
  });
}

// Événement pour un bouton .translate-softskills-btn (facultatif)
$(document).on('click', '.translate-softskills-btn', function() {
  const activityId = $(this).data('activity-id');
  openTranslateSoftskillsModal(activityId);
});
```

3. Structure de la Base de Données (Texte)

Table: activities

- id (INTEGER)
- shape_id (VARCHAR(50))
- name (VARCHAR(200))
- description (TEXT)
- is_result (BOOLEAN)

Table: data

- id (INTEGER)
- shape_id (VARCHAR(50))
- name (VARCHAR(255))
- type (VARCHAR(50))
- description (TEXT)
- layer (VARCHAR(50))

Table: roles

- id (INTEGER)
- name (VARCHAR(100))

Table: tools

- id (INTEGER)
- name (VARCHAR(255))
- description (TEXT)

Table: activity_roles

- activity_id (INTEGER)
- role_id (INTEGER)
- status (VARCHAR(50))

Table: competencies

- id (INTEGER)
- description (TEXT)

• activity_id (INTEGER)

Table: links

• id (INTEGER)

• source_activity_id (INTEGER)

• source_data_id (INTEGER)

• target_activity_id (INTEGER)

• target_data_id (INTEGER)

• type (VARCHAR(50))

• description (TEXT)

Table: softskills

• id (INTEGER)

• habilete (VARCHAR(255))

• niveau (VARCHAR(10))

• activity_id (INTEGER)

Table: tasks

• id (INTEGER)

• name (VARCHAR(255))

• description (TEXT)

• order (INTEGER)

• activity_id (INTEGER)

Table: task_roles

• task_id (INTEGER)

• role_id (INTEGER)

• status (VARCHAR(50))

Table: task_tools

• task_id (INTEGER)

• tool_id (INTEGER)

4. Diagramme Visuel de la Base de Données

Diagramme généré : C:\Users\Hubert.AFDEC\OneDrive - A.F.D.E.C\Documents\DevOPTIQ_Recup\ProjetOPTIQ\database_schema.png