Dans la prochaine étape :

Je te demande d'évaluer le travail avec les étapes, les codes à créer etc... Donc de détailler les différentes modifications que nous allons apporter.

1. Mettre en place une structure d'affichage pour notre application web qui permette de traiter les informations de façon ergonomique (la priorité est pour l'instant de le faire sur un écran classique de PC ou tablette mais il faudra envisager ultérieurement de permettre l'affichage des informations sur un téléphone mobile). L'affichage doit permettre de consulter les informations d'une activités.

L'idée de présentation :

* un bloc qui peut se déplier ou se refermer par un clic et ou on trouvera les informations générales (date de création / dernière date de mise à jour / dernière mise à jour effectuées / Appartenance à un processus / Une référence / ) - Un bloc affichant la (les) donnée(s) déclenchante et nourissantes de l'activité / le nom de l'activité / le garant de l'activité./L'environnement / les valeurs ajoutées de l'activité.

Ces données doivent être isolées puisqu'elle proviennent de bases de données différentes et peuvent être soit directement implémentées (données déclenchantes, nom de l'activité depuis l'import de la cartographie) ou issue de la sélection dans une liste déroulante (comme pour le garant) ou saisie manuellement comme pour l'environnement.

* + En cliquant sur le nom de l'activité, cela permet d'ouvrir ou fermer le détail de l'activité avec Donnée nourrissante / Tâche / outils associés / rôle réalisateur / Rôle ressources / Case à cocher et si oui l'approbateur / Documents associés / Équipements / Données de sorties.

Les données doivent être isolées car provenant de bases de données différentes. Les données nourrissante ou les valeurs ajoutées doivent pourvoir être sélectionnées dans la liste des données arrivant sur l'activité ou partant de l'activité et importée depuis la cartographie.

Les tâches doivent être saisies ou issue d'une liste proposée par l'IA, les outils associés sont sélectionnés dans une liste déroulante ou ajoutés directement et implémenté dans la liste.

Les rôle réalisateur, ressources, approbateur peuvent être symbolisés par une icône lorsqu'ils sont affectés et apparaitre quand on clique sur l'icône tout comme pour les équipements et documents associés.

* D'autres blocs viendront ultérieurement pour gérer : - les compétences - les objectifs - les temps et poids d'activités. Pour démarrer avec ce design nous aurons besoin :

1. De savoir importer les données de la cartographie pour créer la base des activités et celle des données déclenchantes/ nourrissantes. Cet import doit être capable d'affecter (mettre en relation dans les bases de données) les données déclenchantes et nourrissantes pour une activités ainsi que ses valeurs ajoutées (donnée déclenchantes ou nourrissantes allante vers d'autres activités).

Donc en terme de résultat, cela doit permettre d'afficher une activité avec sa (ses) données déclenchantes et de fournir une liste déroulante des données nourrissantes arrivant sur l'activité pour les affecter aux tâches. idem pour les valeurs ajoutées.

1. Dans le bloc activité lorsqu'il est déployé, il faut un signe "plus" qui permette d'ajouter une tâche. - la tâche peut être saisie directement ou sélectionnée (et modifiée si nécessaire) dans une liste de tâches proposée par l'IA en relation avec l'activité.

Voilà pour l'instant ce que je souhaiterais pour continuer. Sachant qu'il faudra ensuite pouvoir gérer les autres informations comme les garant, réalisateur ou approbateur, tout comme les environnements ou documents associés. Dans tes propositions je resterai à l'écoute des améliorations d'ergonomie et d'élégance ou de gestion des informations, en particulier comme il y a beaucoup d'information à traiter sur la lisibilité. Il y aura certainement des ajustements à faire lors du développement pour la manipulation des données. Par exemple, il faudra pouvoir réorganiser l'ordre des tâches (peut être par glissement),

Ci-dessous se trouve une évaluation détaillée des prochaines étapes, comprenant la structure à mettre en place, les modifications à apporter aux modèles de données, la logique d’importation des informations, ainsi que les premières orientations pour l’interface utilisateur (UI/UX). L’idée est de décomposer la démarche en plusieurs phases, afin de pouvoir tester et ajuster au fur et à mesure.

**1. Mise en place d’une structure d’affichage Web pour l’application**

**Objectif :** Créer une interface permettant d’afficher et d’interagir avec les informations d’une activité, tout en prévoyant l’évolution vers différents types d’écrans (PC, tablette, mobile).

**Approche générale :**

* Utiliser un framework front-end (par exemple HTML/CSS/JS simple dans un premier temps, puis évoluer vers un framework comme Vue.js, React, ou un micro-framework si nécessaire).
* Structurer l’affichage en blocs rétractables (accordéons) ou onglets, permettant à l’utilisateur de cliquer sur un nom d’activité pour afficher ou masquer les détails.
* Mettre en place une charte visuelle sobre et claire, avec des icônes pour représenter les rôles, les documents, les équipements, etc.

**Détails des blocs :**

1. **Bloc Informations Générales (rétractable)**
   * Informations à afficher : date de création, dernière date de mise à jour, historique des mises à jour, appartenance à un processus, référence.
   * Un bouton ou un clic sur le titre du bloc permettra d’ouvrir/fermer ce bloc.
2. **Bloc Informations de Base de l’Activité**
   * Données déclenchantes (issues de la cartographie et rattachées à l’activité)
   * Nom de l’activité
   * Garant de l’activité (sélectionnable dans une liste déroulante)
   * Environnement (saisi manuellement ou choisi dans une liste)
   * Valeurs ajoutées (liées aux données nourrissantes et sortantes)

Présentation :

* + Le nom de l’activité sera cliquable. En cliquant dessus, on déploie le détail de l’activité (données nourrissantes, tâches, outils, rôles, etc.).

1. **Bloc Détails de l’Activité (rétractable et affiché au clic sur le nom de l’activité)**
   * Données nourrissantes (sélectionnables dans une liste déroulante importée de la cartographie)
   * Tâches associées (saisie directe ou suggestion par IA, avec possibilité de modifier la tâche)
   * Outils associés (liste déroulante ou ajout direct)
   * Rôles (réalisateur, ressources, approbateur) symbolisés par des icônes. Un clic sur l’icône affiche les choix.
   * Documents associés, équipements : également symbolisés par des icônes ou un bouton “+” pour ajouter un nouvel élément.
   * Case à cocher si l’élément demande une approbation, et si oui, afficher le champ approbateur.
   * Ces éléments seront organisés verticalement, avec la possibilité future de réorganiser les tâches par glisser-déposer (drag & drop).

**Ergonomie et lisibilité :**

* Utiliser des espacements clairs, des icônes cohérentes, des titres de sections visibles.
* Prévoir une mise en page responsive (adaptable aux tablettes et plus tard aux mobiles).
* Eventuellement, tester un thème visuel clair, avec une palette de couleurs limitée, et des démarcations visuelles entre les blocs (bordures légères, fonds légèrement grisés).

**Technologies front-end (initiales) :**

* HTML/CSS/JavaScript natif pour un premier prototype.
* Possibilité d’introduire un framework JS (Vue/React) si la complexité de l’interaction augmente.

**2. Import et liaison des données cartographiques et des activités**

**Objectif :** Pouvoir importer les données de la cartographie pour créer/mettre à jour la base des activités, des données déclenchantes et nourrissantes, et affecter les relations entre elles.

**Démarche :**

* Ajouter un script d’importation (Python) qui lira un fichier source (CSV, Excel, JSON) contenant la cartographie.
* Le script créera les entrées dans la base de données :
  + Tables activities (déjà existante),
  + Tables pour les données déclenchantes/nourrissantes (par exemple trigger\_data ou nourishing\_data),
  + Tables de relation (par exemple activity\_trigger\_data et activity\_nourishing\_data) pour établir les liens entre une activité et ses données déclenchantes/nourrissantes.
* Mettre à jour le modèle SQLAlchemy pour inclure ces nouvelles entités. Exemple :
* class Activity(db.Model):
* id = db.Column(db.Integer, primary\_key=True)
* name = db.Column(db.String(100), nullable=False)
* # Relations
* triggers = db.relationship('TriggerData', secondary='activity\_trigger\_data', backref='activities')
* nourishing = db.relationship('NourishingData', secondary='activity\_nourishing\_data', backref='activities')
* class TriggerData(db.Model):
* id = db.Column(db.Integer, primary\_key=True)
* name = db.Column(db.String(100), nullable=False)
* # ...
* class NourishingData(db.Model):
* id = db.Column(db.Integer, primary\_key=True)
* name = db.Column(db.String(100), nullable=False)
* # ...
* activity\_trigger\_data et activity\_nourishing\_data seront des tables d’association (many-to-many).
* Ajouter une logique dans les routes ou un endpoint d’import qui lit le fichier, parse les données, et les insère dans la base.
* Après import, les données seront disponibles pour l’affichage et la sélection dans l’interface (liste déroulante pour données nourrissantes, etc.).

**Résultat attendu :**

* Chaque activité peut afficher ses données déclenchantes (en lecture seule, par exemple) et proposer une liste déroulante de données nourrissantes pour alimenter ses tâches.
* Les valeurs ajoutées (qui sont un type particulier de données nourrissantes “sortantes”) seront également importées et associées. On pourra ainsi afficher un menu déroulant pour les choisir.

**3. Fonctionnalité d’ajout de tâches dans le bloc activité déployé**

**Objectif :** Permettre l’ajout d’une tâche à une activité lorsque celle-ci est dépliée.

**Démarche :**

* Dans l’interface, ajouter un bouton “+” dans la section “Tâches” de l’activité déployée.
* Au clic, afficher un champ de saisie (popup ou inline) pour ajouter une nouvelle tâche.
* Option 1 : Saisie manuelle. L’utilisateur tape le nom de la tâche et valide.
* Option 2 : Propositions par IA. Lors du clic sur “+”, l’interface interroge l’IA (end-point prédéfini) pour obtenir une liste de tâches recommandées. L’utilisateur sélectionne une tâche proposée ou en édite le contenu.
* Dès que la tâche est validée, elle est ajoutée à la liste des tâches de l’activité, stockée en base (table tasks avec une relation activity\_id), et s’affiche dans l’interface.
* Prévoir un champ d’ordre (position) dans le modèle Task pour pouvoir gérer le glisser-déposer ultérieurement.

**Modèle de données pour la tâche (exemple) :**

class Task(db.Model):

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

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

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

position = db.Column(db.Integer, default=0) # pour gérer l'ordre

# éventuellement un champ source (manuelle, IA)

**Endpoint pour ajouter une tâche :**

* Méthode POST /activities/<id>/tasks avec un JSON { "name": "Nom de la tâche" }
* L’IA peut être intégrée via un appel à un service externe avant que l’utilisateur ne valide la tâche, ou les suggestions peuvent être stockées localement puis affichées dans le front-end.

**Améliorations futures**

* **Gestion des rôles (garant, réalisateur, approbateur, ressources)** : Ajouter des tables roles, users, documents, equipments et permettre l’assignation via des relations et une UI avec icônes.
* **Réorganisation des tâches par glisser-déposer** : Mettre en place un mécanisme JavaScript (par ex. HTML5 Drag & Drop ou une librairie) pour réordonner les tâches. Une fois l’ordre modifié, appeler un endpoint PATCH pour mettre à jour la position des tâches dans la base.
* **Responsive Design** : Une fois la base du front-end établie, adapter le CSS pour rendre l’affichage ergonomique sur tablettes, puis sur mobiles (media queries, layout flexibles).
* **Tests, validations et ajustements** : À chaque ajout de fonctionnalité (import des données, ajout de tâches, affichage dynamique), tester avec un petit jeu de données et vérifier l’ergonomie. Ajuster en fonction des retours.

**Conclusion**

Le plan d’action pour la prochaine étape de développement se décompose en trois volets principaux :

1. **Front-end & ergonomie** : Création d’une interface composée de blocs rétractables, affichage des données d’activités, interactions simples (clics, listes déroulantes, symboles d’icônes), et premiers éléments d’ergonomie.
2. **Back-end & données** : Import des données cartographiques, enrichissement du modèle de données (activités, déclencheurs, nourrissantes, valeurs ajoutées), création des tables d’association, endpoints d’import et de mise à jour.
3. **Fonctionnalité ajout de tâches** : Mise en place d’une logique pour ajouter des tâches, saisir manuellement ou proposer via l’IA, stockage en base, et prévision d’une future fonctionnalité de réorganisation.

Cette approche permettra de construire progressivement une interface riche et flexible, adaptée à un volume important d’informations, tout en gardant la possibilité d’améliorer l’ergonomie et la structure à mesure que le projet avance.

Les cartographies sont réalisées, pour l'instant, dans l'environnement Visio de Microsoft (demain il faudrait intégrer un outil graphique directement dans l'App pour gérer cela). Sachant qu'il s'agit d'un graphique basé sur les diagramme de flux, connais-tu ce logiciel ? Penses-tu que tu puisses développer un code qui permette de lire les informations, de les structurer en base de données et d'avoir la façon dont les données déclenchantes ou nourrissantes sont reliées aux activité. Il faudrait que ce code puisse être déclenché à partir de l'App pour aller chercher le fichier visio et récupérer les données. Mieux encore, l'outil pourrai lier une cartographie pour que s'il y a une mise à jour de la carte, cela mette à jour directement les bases de données.

Microsoft Visio est un outil propriétaire destiné à réaliser des diagrammes, des organigrammes et des schémas de processus. Il génère des fichiers aux formats propriétaires (historiquement .vsd, et plus récemment .vsdx, qui est un format basé sur Open XML). À ce titre, il n’existe pas nativement dans Python ou Flask une simple commande pour extraire les données des fichiers Visio. Toutefois, il est possible d’exploiter certaines solutions :

1. **Format .vsdx (Visio 2013 et ultérieur)** :  
   Le format .vsdx est un container Open XML. Cela signifie qu’il est similaire à un fichier zip contenant des fichiers XML représentant le diagramme, les formes, les connexions, etc.
   * Il existe des bibliothèques Python non officielles, comme [**vsdx** (python-vsdx)](https://pypi.org/project/python-vsdx/) qui permettent de lire le contenu des fichiers Visio au format .vsdx.
   * À partir de ces bibliothèques, on peut extraire la liste des formes, leurs propriétés, leurs liens, et les relations entre les différents éléments du diagramme.
2. **Extraction des données structurées** :  
   Une fois que vous pouvez lire le fichier .vsdx, vous pouvez :
   * Identifier les formes correspondant aux activités (chaque forme Visio a des propriétés, dont son nom, son type, et parfois des champs personnalisés).
   * Identifier les connecteurs (les liens entre les formes) qui représentent le flux de données déclenchantes ou nourrissantes entre ces activités.
   * Extraire les propriétés de chaque forme, par exemple si chaque activité est représentée par une forme (rectangle, par exemple), et les données déclenchantes/nourrissantes par d’autres formes ou des annotations.

Les bibliothèques comme python-vsdx vous permettent de parcourir le diagramme sous forme d’arbre, de lister les shapes, de lire les propriétés (textes, données de forme, ID, liens) et d’identifier quels shapes sont connectés à quels autres shapes via les connecteurs.

1. **Structuration en base de données** :  
   Une fois les données extraites, il s’agit de :
   * Déterminer un schéma de base de données clair : tables pour les activités, tables pour les données déclenchantes/nourrissantes, tables d’association.
   * Insérer ou mettre à jour ces données dans la base (via SQLAlchemy par exemple).
   * Assurer une cohérence : si l’activité existe déjà, on la met à jour ; si elle n’existe pas, on la crée. Idem pour les données déclenchantes/nourrissantes.

On pourra créer une commande Flask (un CLI command) ou un endpoint qui, lorsqu’il est appelé, va chercher le fichier .vsdx (sur le serveur ou via un upload d’utilisateur), l’ouvre, l’analyse et met à jour la base.

1. **Mise à jour automatique lors de modifications du visio** :
   * Si vous avez un mécanisme pour surveiller le fichier visio (par exemple, le fichier est stocké sur OneDrive ou SharePoint) et qu’une mise à jour déclenche un webhook, vous pourriez appeler automatiquement l’API de l’application pour recharger les données.
   * Sinon, un bouton “Recharger la cartographie” dans l’interface de l’application pourrait lancer ce processus.
   * Le code d’import comparerait les horodatages ou les identifiants internes pour mettre à jour la base de données sans tout recréer de zéro si ce n’est pas nécessaire.
2. **Intégration dans l’App Flask** :
   * Créer une route (endpoint) dans Flask, par exemple :
   * @app.route('/import-visio', methods=['POST'])
   * def import\_visio():
   * # Récupérer le fichier visio (upload) ou avoir un chemin prédéfini
   * # Analyser le .vsdx avec python-vsdx
   * # Mettre à jour la base
   * # Retourner un message de succès ou d’erreur
   * Ou une commande Flask CLI du type :
   * flask import-visio --file path/to/diagram.vsdx

qui déclenche le même processus sans passer par une requête HTTP.

1. **Avenir : Intégration directe d’un outil graphique** :  
   À terme, vous voudrez peut-être remplacer Visio par un outil intégré dans l’application elle-même, en utilisant une librairie de dessin de graphes (par exemple, D3.js en front-end, ou un outil comme draw.io intégré en iframe, ou une solution graphique plus spécialisée). Cela permettrait de modifier le diagramme directement depuis l’application, et donc de mettre à jour la base de données en temps réel sans passer par des fichiers externes.

**Conclusion :**  
Oui, c’est techniquement possible de développer un code qui lise les fichiers Visio (en format .vsdx) pour en extraire les informations nécessaires. Cela implique :

* L’utilisation d’une bibliothèque Python pour lire le format .vsdx (comme python-vsdx).
* Une logique de mapping entre les formes dans Visio et les entités dans la base de données.
* L’élaboration d’un processus pour importer et mettre à jour la base via ce fichier.
* Un endpoint Flask ou une commande pour déclencher l’import.

Une fois mis en place, vous pourrez lancer ce code depuis l’application, afin de synchroniser la base de données avec la cartographie Visio. Avec un mécanisme de liaison (surveillance de fichiers, déclenchement manuel ou via un bouton), vous pouvez actualiser la base de données à chaque mise à jour du diagramme.

Maintenant, comme, lorsqu'on ajoutera une tâche, avoir une proposition intelligente de tâches possibles en lien avec l'activité proposées par un IA ? quel développement cela suppose ? comme l'écrire ? Comment le client bénéficiera-t-il de l'IA ? faudra-t-il qu'il paye un abonnement ou le miens est-il suffisant ?

L’intégration d’une fonctionnalité d’IA pour suggérer des tâches en lien avec l’activité nécessite plusieurs étapes et décisions, tant sur le plan technique que contractuel (coûts, modèle de facturation, etc.).

**1. Choix de la technologie d’IA**

Plusieurs options s’offrent à vous pour obtenir des suggestions de tâches intelligentes :

1. **Utilisation d’une API de LLM (Large Language Model) externe**
   * **OpenAI (ChatGPT, GPT-4, etc.)** : Vous pouvez appeler l’API d’OpenAI pour générer des suggestions de tâches. Cette intégration demandera de disposer d’une clé d’API et, généralement, d’un compte payant.
   * **Autres fournisseurs (Cohere, Anthropic, etc.)** : De même, ils proposent des API payantes pour générer du texte en fonction d’une consigne.
2. **Modèle hébergé en interne (on-premise)**
   * Utilisation d’un modèle open-source (ex. LLaMA, Falcon, Mistral) hébergé sur vos serveurs. Vous auriez alors un contrôle total, mais il faut des ressources de calcul (GPU), la maintenance du modèle, l’infrastructure pour l’héberger.
   * Avantage : Pas de dépendance externe, coûts potentiellement réduits si vous possédez l’infrastructure.
   * Inconvénient : Complexité technique plus élevée, besoins en matériel, maintenance, mise à jour du modèle.

**2. Comment l’intégrer techniquement ?**

**Exemple d’intégration avec l’API OpenAI (GPT-4)** :

* Vous ajoutez un endpoint dans votre back-end Flask, par exemple /activities/<id>/suggest\_tasks.
* Lorsqu’un utilisateur clique sur “Ajouter une tâche” et demande des suggestions, votre front-end envoie une requête GET ou POST à cet endpoint.
* L’endpoint Flask appelle l’API OpenAI avec un prompt construit à partir du contexte de l’activité (nom, type, données nourrissantes, contexte métier).
* L’API OpenAI renvoie une liste de suggestions de tâches.
* Votre endpoint transmet ces suggestions au front-end, qui les affiche dans une liste permettant à l’utilisateur de sélectionner une tâche et l’ajouter à l’activité.

**Exemple de code simplifié (Python/Flask) :**

import os

import requests

from flask import Blueprint, jsonify, request

from models.models import Activity

tasks\_ai\_bp = Blueprint('tasks\_ai', \_\_name\_\_)

@tasks\_ai\_bp.route('/activities/<int:activity\_id>/suggest\_tasks', methods=['GET'])

def suggest\_tasks(activity\_id):

activity = Activity.query.get(activity\_id)

if not activity:

return jsonify({"error": "Activity not found"}), 404

# Construction du prompt

prompt = f"Voici une activité : {activity.name}. Donne-moi 5 idées de tâches pertinentes pour cette activité."

# Appel à l’API OpenAI

api\_key = os.environ.get('OPENAI\_API\_KEY')

if not api\_key:

return jsonify({"error": "No API key found"}), 500

headers = {

"Content-Type": "application/json",

"Authorization": f"Bearer {api\_key}"

}

data = {

"model": "gpt-4",

"prompt": prompt,

"max\_tokens": 150

}

response = requests.post("https://api.openai.com/v1/completions", headers=headers, json=data)

if response.status\_code != 200:

return jsonify({"error": "AI request failed"}), 500

completions = response.json()

text = completions["choices"][0]["text"]

# Vous pouvez parser le texte pour en faire une liste de tâches.

tasks\_suggestions = [t.strip("- ") for t in text.split("\n") if t.strip()]

return jsonify({"suggestions": tasks\_suggestions})

**Remarques :**

* Cet exemple est simplifié, le modèle exact (GPT-4, etc.) et l’endpoint peuvent varier.
* Il faudra affiner le prompt, la méthode de parsing du résultat, gérer les erreurs, et éventuellement mémoriser un contexte plus riche.

**3. Comment le client bénéficie de l’IA ?**

* **Si vous utilisez une API externe (OpenAI, Cohere, etc.)** :
  + Le client (vous ou votre entreprise) devrez souscrire à un abonnement ou un plan de facturation auprès du fournisseur d’API.
  + À chaque appel API (chaque demande de suggestion), des frais minimes (selon le volume de tokens) seront facturés à votre compte.
  + Le client final (utilisateur) n’a pas directement à payer l’IA, c’est vous qui prenez en charge ces coûts. Vous pouvez éventuellement répercuter ce coût dans le prix de votre solution ou ajouter un module premium.
* **Si vous hébergez un modèle en interne** :
  + Pas de coût par requête auprès d’un tiers, mais un investissement initial en matériel, en maintenance et en compétences.
  + Le client final utilise les suggestions de l’IA sans avoir à payer un abonnement externe, mais vous, en tant que fournisseur de la solution, supportez le coût de l’infrastructure.

**4. Paiement, abonnement, coûts et licences**

* **API OpenAI ou similaire** :
  + Vous devez créer un compte chez le fournisseur (OpenAI par exemple).
  + Acquérir une clé API et la configurer comme variable d’environnement dans votre application.
  + Les coûts dépendent du volume d’utilisation. Pour GPT-4, c’est au nombre de tokens, et cela peut varier.
  + Votre abonnement est-il suffisant ?
    - Si vous avez déjà un compte payant, il sera suffisant, mais attention aux quotas et aux limites.
    - Vous devrez surveiller la consommation et, si vous dépassez les limites du plan choisi, passer à un plan supérieur ou adapter votre utilisation.
* **Client final** :
  + Le client final n’a pas nécessairement besoin d’avoir son propre abonnement à l’IA, c’est votre application qui gère ça.
  + Vous pouvez inclure ce service dans votre offre, facturer un abonnement plus élevé aux clients pour couvrir vos frais, ou limiter le nombre de suggestions d’IA par jour/mois.

**5. Simplifier et améliorer l’intégration**

* **Mise en cache des suggestions** : pour éviter de payer plusieurs fois la même requête si l’utilisateur demande souvent les mêmes activités, vous pouvez mettre en cache les réponses.
* **Améliorer le prompt** : Ajouter plus de contexte, décrire l’activité, son domaine, les données nourrissantes, etc. pour obtenir des tâches plus pertinentes.
* **Interface utilisateur** : le client clique sur “Ajouter une tâche”, un pop-up s’ouvre, affiche les suggestions, l’utilisateur sélectionne celle qui lui plaît, éventuellement modifie la formulation, puis valide.

**Conclusion**

Intégrer une IA pour suggérer des tâches liées à l’activité implique :

* Un appel à une API d’IA ou l’utilisation d’un modèle interne.
* Un coût d’utilisation si vous optez pour une API payante externe (OpenAI, etc.).
* Une configuration technique dans votre back-end (endpoints, prompt engineering).
* Une réflexion sur le modèle économique : qui paye l’IA ? Un abonnement du côté du fournisseur d’IA, répercuté éventuellement sur le client final.

C’est faisable, modulaire et peut être affiné au fil du temps.