**Documentation CHATBOT**

**🎯 À quoi sert le contexte exactement dans ton chatbot ?**

Le **contexte** est un résumé intelligent qui regroupe des informations pertinentes sur l'utilisateur et les récents échanges de conversation, afin d'améliorer la pertinence et la précision des réponses fournies par le chatbot.

**🧠 Objectifs principaux du contexte :**

1. **Personnaliser les réponses :**  
   En tenant compte du profil utilisateur (objectifs, compétences, attentes, etc.), le chatbot peut adapter ses réponses aux besoins spécifiques de chaque personne.
2. **Améliorer la cohérence conversationnelle :**  
   En transmettant les derniers échanges (questions et réponses précédentes), le chatbot évite les réponses génériques ou hors contexte. Il se souvient en quelque sorte des échanges précédents.
3. **Optimiser la recommandation de formations :**  
   Grâce aux données de profil et à l'historique, le chatbot propose des recommandations de formation plus précises et ciblées.

**📦 Exemple concret :**

Imaginons un utilisateur avec le profil suivant :

json

CopierModifier

{

"nom": "Jean Dupont",

"situation\_actuelle": "en reconversion",

"objectif": "devenir Data Analyst",

"niveau\_actuel": "débutant",

"connaissances": ["Excel", "HTML"],

"attentes": "apprendre Python, maîtriser Power BI"

}

Si l'utilisateur demande :

**« Quelle formation me conseillerais-tu ? »**

Sans contexte, le chatbot pourrait répondre vaguement :

« Voici une liste générale de formations disponibles… »

Avec le contexte enrichi (profil utilisateur), le chatbot répondra précisément :

« Étant donné votre objectif de reconversion en Data Analyst et votre souhait d’apprendre Python, voici une formation débutant parfaitement adaptée à vos attentes… »

**🛠️ Contenu typique du contexte envoyé au backend :**

* ✅ **Profil utilisateur :**

text

CopierModifier

Profil utilisateur :

- Nom : Jean Dupont

- Situation actuelle : en reconversion

- Objectif : devenir Data Analyst

- Niveau actuel : débutant

- Connaissances : Excel, HTML

- Attentes : apprendre Python, maîtriser Power BI

* ✅ **Historique conversationnel récent :**

text

CopierModifier

Utilisateur : Je cherche une formation en Python.

Assistant : Souhaitez-vous une formation spécifique orientée data science ou plutôt développement web ?

Utilisateur : Plutôt Data Science.

Ces données combinées permettent au modèle de :

* Comprendre clairement la requête.
* Éviter de poser des questions déjà clarifiées.
* Fournir des réponses personnalisées et pertinentes immédiatement.

**🚀 En résumé :**

Le contexte sert donc principalement à enrichir les échanges, augmenter la pertinence des réponses du chatbot et offrir une expérience utilisateur réellement adaptée et qualitative.

**. Avoir un prototype abouti, fluide et stable**

* **Frontend** :
  + Vérifier que le formulaire de profil est **sans bug** et **complet** (aucun champ vide qui plante, validations minimales).
  + S’assurer que le chat fonctionne **sans latence excessive**, et que l’historique est **proprement affiché**.
  + Confirmer que le **bouton de recommandation** s’affiche toujours quand il le faut et **retourne quelque chose** (même un message d’erreur clair si rien n’est trouvé).
* **Backend** :
  + Vérifier **aucune erreur CORS** ni 422 Unprocessable Entity dans les use cases standards.
  + Vérifier que les endpoints /query et /recommend **répondent** même en cas de données incohérentes (logique de fallback).

Ces éléments donnent confiance dans la solidité du prototype.

**2. Clarifier la logique de recommandation**

* **Documenter** (en quelques lignes) comment se fait le **match** entre le profil et la formation.
* Mentionner dans la présentation si la logique est **simple** (matching sur mots-clés / publics) ou plus **complexe** (avec scoring).
* Pour le chef de projet, c’est rassurant de voir **comment** la solution choisit une formation.

**3. Mettre en avant la gestion du contexte**

* Expliquer (avec un mini diagramme / workflow) comment le contexte conversationnel est transmis du frontend au backend, et **pourquoi** il aide à améliorer la pertinence.
* Souligner que c’est **prêt** pour l’intégration LLM, c’est un point fort à valoriser.

**4. Fournir un mini “Guide d’utilisation” interne**

* **Pour un QA rapide** : quelques steps (“1. Lancer uvicorn, 2. Lancer ng serve, 3. Se rendre sur <http://localhost:4200>, 4. Tester un profil”).
* **Pour le mainteneur** : où se trouvent les fichiers clé (backend/main.py, frontend ChatComponent).

**5. Prochaine étape : Intégration LLM par Mohammed**

* Insister sur le fait que la structure est **prête** pour la partie LLM (context, prompts).
* Montrer que **le point d’accroche est clair** : l’endpoint /query côté backend, et la méthode sendMessage() côté frontend.

**6. Soigner la démo**

* Préparer un **exemple concret** (profil + 2-3 questions) pour le chef de projet :
  + “Jean Dupont, en reconversion Data Analyst, souhaite apprendre Python”
  + **Démonstration** : en live, montrer le remplissage du profil, la réponse du chatbot, la recommandation, etc.

De cette façon, tu montres un **produit fonctionnel**, avec une logique claire et une vision de la suite (LLM). Ton chef de projet appréciera la **transparence** (documentation rapide), la **fiabilité** (pas de bug visible) et la **perspective** (comment Mohammed va brancher son LLM).

**Conclusion**

Focus sur la **stabilité**, la **clarté** (documentation/logique) et la **démonstration**. Ton chef de projet verra un **projet solide**, déjà apte à répondre aux besoins, et prêt à évoluer. Bonne chance pour ta présentation !

**Vous avez dit :**

Peux-tu mettre en place tous ces éléments ?

**ChatGPT a dit :**

Raisonnement terminé pendant 4 secondes

Très bien ! Rassemblons et consolidons tous les points pour avoir un **prototype fiable**, **bien documenté**, **stable**, et prêt pour la présentation. Je te propose ci-dessous :

1. **Un récapitulatif du code** (côté frontend et backend) déjà stabilisé,
2. **Un mini guide d’utilisation interne** (pour QA rapide),
3. **Une logique de matching un peu plus robuste** (avec fallback en cas d’erreur),
4. **Une courte documentation** que tu pourras insérer dans ton README ou slides,
5. **Un exemple concret** (pour la démo au chef de projet).

De cette façon, tu as un package complet prêt à être présenté.

**1. Code stabilisé : Backend (main.py)**

Ci-dessous, un code qui tient compte de tous les retours, avec un **fallback** s’il y a une erreur interne, et un matching plus clair :

python

CopierModifier

from fastapi import FastAPI, HTTPException

from fastapi.middleware.cors import CORSMiddleware

from pydantic import BaseModel

from typing import List, Optional

import json

from pathlib import Path

app = FastAPI()

# Configuration CORS pour autoriser Angular

app.add\_middleware(

CORSMiddleware,

allow\_origins=["http://localhost:4200"],

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

# Chargement des formations JSON

json\_dir = Path("content/json\_clean/formations")

formations = []

try:

for file in json\_dir.glob("\*.json"):

with open(file, "r") as f:

data = json.load(f)

formations.append(data)

except Exception as e:

print(f"[ERREUR] Impossible de charger les formations : {e}")

# -------------------------------------

# Modèles

# -------------------------------------

class UserProfile(BaseModel):

nom: str

situation\_actuelle: str

objectif: str

niveau\_actuel: str

connaissances: List[str]

attentes: str

class MessageContext(BaseModel):

sender: str

text: str

class ChatContext(BaseModel):

prompt: str

role: Optional[str] = "user"

context: Optional[str] = None

class ChatContextResponse(BaseModel):

response: str

source\_documents: Optional[List[str]] = None

class RecommendationResponse(BaseModel):

recommandation: str

details: Optional[dict] = None

formation\_link: Optional[str] = None

message: Optional[str] = None

# -------------------------------------

# Endpoint principal /query

# -------------------------------------

@app.post("/query", response\_model=ChatContextResponse)

def query(chat: ChatContext):

"""

Reçoit un 'prompt', un 'role' et un 'context' (historique + profil).

Retourne une réponse simulée pour le moment.

"""

prompt = chat.prompt

role = chat.role

context = chat.context

# Log du contexte reçu

print("\n[DEBUG] /query - Contexte reçu :")

print(context)

# Simulation d'une réponse (Mohammed branchera son LLM ici)

fake\_response = f"Réponse simulée pour : '{prompt}' (role={role}), tenant compte du contexte."

fake\_sources = [

"Formation\_Python\_Data\_Visualisation.json",

"Formation\_Power\_BI.json"

]

return ChatContextResponse(response=fake\_response, source\_documents=fake\_sources)

# -------------------------------------

# Logique de Recommandation

# -------------------------------------

@app.post("/recommend", response\_model=RecommendationResponse)

def recommend(profile: UserProfile):

"""

Reçoit un profil utilisateur (POST),

et renvoie la formation la plus adaptée ou un fallback.

"""

criteres = extraire\_criteres(profile)

formation = match\_formation(criteres, formations)

if formation:

return RecommendationResponse(

recommandation=formation["titre"],

details={

"objectifs": formation.get("objectifs", []),

"public": formation.get("public", [])

},

formation\_link=formation.get("lien", "#")

)

else:

return RecommendationResponse(

recommandation="Aucune formation pertinente trouvée.",

message="Proposer un bilan de compétences.",

formation\_link=None

)

# -------------------------------------

# Fonctions utilitaires

# -------------------------------------

def extraire\_criteres(profil: UserProfile) -> dict:

"""

Extrait les critères utiles du profil :

- objectifs

- compétences (niveau + connaissances)

- public

"""

objectifs = [profil.objectif] + profil.attentes.split(",")

objectifs = [o.strip().lower() for o in objectifs]

competences = [profil.niveau\_actuel.lower()] + [c.lower() for c in profil.connaissances]

# Public 'étudiants' ou 'tout public'

public = ["étudiants"] if "étudiant" in profil.situation\_actuelle.lower() else ["tout public"]

return {

"objectifs": objectifs,

"competences": competences,

"public": public

}

def match\_formation(criteres: dict, formations: list) -> Optional[dict]:

"""

Recherche la première formation qui correspond

aux objectifs, prérequis ou public.

"""

for formation in formations:

# On fait un to-lower sur la formation pour comparer

formation\_objectifs = [obj.lower() for obj in formation.get("objectifs", [])]

formation\_prerequis = [pr.lower() for pr in formation.get("prerequis", [])]

formation\_public = [p.lower() for p in formation.get("public", [])]

# Condition de match

if any(obj in formation\_objectifs for obj in criteres["objectifs"]) \

or any(comp in formation\_prerequis for comp in criteres["competences"]) \

or any(pub in formation\_public for pub in criteres["public"]):

return formation

return None

**Points importants dans ce code :**

1. **Chargement des formations** : avec un try/except pour éviter que le backend plante s’il y a une erreur de lecture JSON.
2. **QueryRequest → remplacé ici par ChatContext** : On accepte prompt, role, context.
3. **ChatContextResponse** : La réponse renvoyée au frontend (Mohammed branchera le LLM).
4. **/recommend** : Recevoir le profil, extraire les critères, matcher la formation (avec un fallback clair).
5. **Comparaisons en minuscules** : évite les incohérences de casse.
6. **Logs** dans la console (pour debug) : [DEBUG] /query - Contexte reçu : ...

**2. Mise à jour du Frontend : chat.component.ts**

Rappels :

* **sendRecommandation()** → envoie le profil à /recommend.
* **sendMessage()** → envoie prompt, role, context à /query.

**3. Mini Guide d’utilisation interne**

Pour un QA rapide :

bash

CopierModifier

1) Cloner le repo

2) Lancer le backend :

cd chatbot/backend

uvicorn main:app --reload

3) Lancer le frontend :

cd chatbot/frontend

ng serve

4) Aller sur http://localhost:4200

- Remplir le profil

- Cliquer “Envoyer” pour poser des questions

- Obtenir une recommandation

5) Contrôler la console du backend pour voir le contexte et debug

**4. Documentation courte**

* **Fichier** : README.md
  + Explication de l’architecture (Frontend Angular + Backend FastAPI + LLM futur).
  + Lien vers la doc plus détaillée (PowerPoint, UML, etc.).
  + Instructions d’installation (uvicorn, ng serve).

**5. Exemple de démo concret**

* **Profil** :
  + Nom : Jean Dupont
  + Situation : en reconversion
  + Objectif : devenir Data Analyst
  + Niveau actuel : Débutant
  + Connaissances : [“Excel”, “HTML”]
  + Attentes : “apprendre Python, maîtriser Power BI”
* **Résultat** :
  + Recommandation : “Formation Python pour débutants”
  + Lien : “https://www.example.com/formation-python-debutants”
* **Chat** :
  + “Quelle est la prochaine étape après avoir appris Python ?”
  + Réponse simulée : “Réponse simulée pour : ‘Quelle est la prochaine étape…’”

En combinant ces éléments, tu as maintenant un **code backend** propre, un **frontend** stable, et la **doc** pour ton chef de projet. Ton chatbot est consolidé, prêt pour la présentation. Bonne continuation !