Diapo à créer :

=== DIAPO 1 ===

Titre : Chatbot LLM + Recommandation de Formations

Sous-titre : Un assistant intelligent pour orienter et conseiller les utilisateurs

Présenté par Michel & Mohammed

=== DIAPO 2 ===

Titre : Objectif et Contexte

- Besoin d’un assistant conversationnel capable de :

> Répondre aux questions des utilisateurs

> Recommander des formations adaptées

> Exploiter un LLM (Logique Mohammed)

- Enjeux :

> Automatiser l’orientation vers la bonne formation

> Personnaliser l’expérience utilisateur

Visuel suggéré : Schéma simple montrant un chatbot, un utilisateur et un répertoire de formations connectés.

=== DIAPO 3 ===

Titre : Architecture Globale

- Frontend Angular : Formulaire de profil, interface conversationnelle

- Backend FastAPI : /recommend, /query, accès aux données

- Base de données : Fichiers JSON (formations)

- LLM : Intégration future par Mohammed

Graphique UML (simplifié) :

[Utilisateur] --(1)--> [Frontend Angular] --(2)--> [Backend FastAPI] --(3)--> [Formations]

=== DIAPO 4 ===

Titre : Fonctionnalités Principales

1. Formulaire Profil (Nom, situation, objectif…)

2. Conversation Chat (échanges multi-turn)

3. Recommandation (endpoint /recommend)

4. Contexte conversationnel (profil + historique)

Visuel : Icônes de formulaire, bulle chat, check-mark recommandation

=== DIAPO 5 ===

Titre : Frontend (Angular)

- ChatComponent : signals, computed, handleProfileSubmit, sendRecommandation

- UserProfileFormComponent : Formulaire de saisie

- SCSS : style du chatbot

Schéma UML minimal :

[ChatComponent] -> [UserProfileFormComponent]

=== DIAPO 6 ===

Titre : Backend (FastAPI)

- Endpoints :

/query : Reçoit prompt, role, context

/recommend : Prend un profil, renvoie formation

- CORS : Autorise http://localhost:4200

- JSON de formations : Lecture + matching

Extrait :

@app.post("/recommend")

def recommend(...):

# Logique de matching...

return {...}

=== DIAPO 7 ===

Titre : Logique de Recommandation

1. extraire\_criteres(profil) -> objectifs, competences, public

2. match\_formation(criteres, formations) -> 1 formation

3. Retour { recommandation, details, formation\_link }

Visuel UML :

UserProfile -> extraire\_criteres() -> match\_formation() -> return formation

=== DIAPO 8 ===

Titre : Gestion du Contexte

- Historique conversationnel (5 derniers messages)

- Profil utilisateur

- Rôle user/assistant

- Transmission au backend via QueryRequest

Exemple de contexte :

Profil utilisateur :

- Nom : Jean Dupont

- Objectif : ...

Historique :

Utilisateur : ...

Assistant : ...

=== DIAPO 9 ===

Titre : Documentation Utilisateurs

- Comment utiliser le chatbot ?

1. Cliquer "Modifier le profil"

2. Renseigner informations (nom, objectif…)

3. "Envoyer" pour poser des questions

4. "Obtenir une recommandation" pour trouver une formation

Visuel : Capture d’écran de l’interface

=== DIAPO 10 ===

Titre : Documentation Développeurs

- Installation :

cd chatbot/backend -> uvicorn main:app --reload

cd chatbot/frontend -> ng serve

- Structure de Projet :

backend/main.py

frontend/src/...

- Extensions futures :

> Intégration LLM

> Base de données plus large

> Deployment

=== DIAPO 11 ===

Titre : Flux de Données Simplifié

Schéma :

[Utilisateur] -> [Frontend Angular] -> [Backend FastAPI] -> [Formations JSON]

Explication :

1. L’utilisateur saisit son profil

2. Recommandation + conversation

3. Intégration LLM future

=== DIAPO 12 ===

Titre : Conclusion & Prochaines Étapes

- Chatbot fonctionnel (profil, recommandations)

- Contexte conversationnel géré

- Intégration LLM par Mohammed

- Roadmap :

1. NLP + RAG

2. Amélioration UX

3. Déploiement

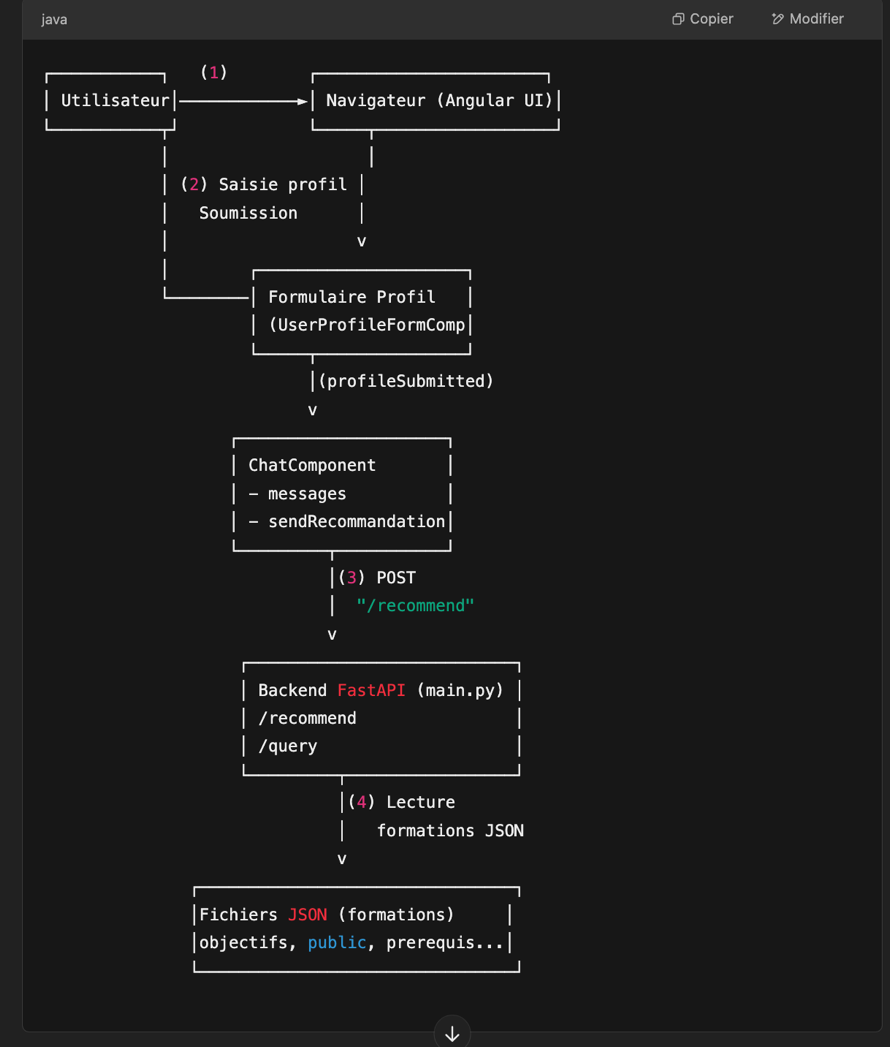
=== DIAPO 13 ===

Titre : Questions ?

"Merci de votre attention !"

Fin

**Explication du schéma :**



1. **(1) Utilisateur** :
   * L’utilisateur interagit avec l'application via son navigateur.
2. **(2) Navigateur (Angular UI)** :
   * Héberge l'interface Angular (<http://localhost:4200>)
   * L’utilisateur y saisit son profil dans un **Formulaire**.
3. **Formulaire Profil** (UserProfileFormComponent) :
   * Au clic sur "Soumettre", déclenche un événement (profileSubmitted).
4. **ChatComponent** :
   * Intercepte (profileSubmitted), stocke le profil, et **envoie** (3) une requête POST /recommend à l’API backend FastAPI.
   * Pour le chat conversationnel, il envoie POST /query (avec un prompt + contexte).
5. **Backend FastAPI (main.py)** :
   * Gère /recommend :
     + Analyse le profil, compare aux **fichiers JSON** (4), et retourne la formation correspondante (ou un message par défaut).
   * Gère /query :
     + Reçoit prompt + contexte, renvoie une réponse simulée (en attendant le LLM de Mohammed).
6. **Fichiers JSON (formations)** :
   * Contiennent la description des formations : objectifs, prérequis, public.
   * Chargés au démarrage du backend, et utilisés pour faire la correspondance profil => formation.

**Flux principaux :**

* **Soumission Profil** :  
  Utilisateur → Formulaire → ChatComponent → /recommend → JSON →
* Réponse formation → ChatComponent → Interface.
* **Chat conversationnel** :  
  Utilisateur → ChatComponent (sendMessage) → /query + contexte → Réponse simulée → ChatComponent → Interface.

**Avantages du schéma ASCII :**

* 100 % lisible depuis n’importe quel environnement.
* Pas de dépendance à un moteur de rendu d’images.
* Facile à modifier/enrichir.