

Un agent IA qui répond à vos clients comme le ferait votre meilleur conseiller — disponible 24h/24, basé uniquement sur vos vrais documents.

Le problème qu'Aria résout

Répondre aux questions clients (produits, garanties, livraison, retours) prend du temps et mobilise une équipe humaine, même pour des questions répétitives. À l'inverse, un chatbot générique invente des réponses ou ne connaît rien à votre activité.

Aria lit vos documents (PDF, fiches produits, FAQ, CGV...) et répond uniquement à partir de ces sources — avec citation du document utilisé. Si l'information n'existe pas, Aria le dit clairement au lieu d'inventer.

Pour qui ?

- **E-commerce / boutiques en ligne** — répondre aux questions produits, livraison, retours sans intervention humaine
- **Services support / SAV** — réduire le volume de tickets répétitifs
- **Toute entreprise avec une base de connaissance** (procédures internes, documentation produit) qui veut la rendre interrogeable en langage naturel

Comment ça marche, concrètement

1. Vous glissez vos documents (PDF, TXT, CSV) dans l'interface
2. Aria les indexe automatiquement
3. Un client pose une question en langage naturel
4. Aria cherche dans vos documents, répond en citant la source, et peut aussi vérifier une info en temps réel sur le web (horaires, actualités) si besoin

Exemple concret :

Client : *"Est-ce que je peux retourner un article après 30 jours ?"* Aria : *"Selon notre politique de retour (CGV.pdf), les retours sont acceptés jusqu'à 14 jours après réception. Passé ce délai, contactez le service client pour étudier votre cas."*

La réponse arrive token par token en temps réel (comme ChatGPT), pas d'attente bloquante.

Stack technique

React/Vite · FastAPI · LangChain (create_agent) · Mistral AI · ChromaDB · PostgreSQL

Architecture full-stack : agent LLM avec tool-calling (recherche documentaire RAG + recherche

web + horloge), mémoire de conversation persistante par session, streaming temps réel via Server-Sent Events.

Structure du projet

```
aria-project/
├── backend/
│   ├── main.py           # API FastAPI + agent LangChain + streaming
│   ├── rag.py            # Indexation et recherche documentaire (ChromaDB)
│   ├── requirements.txt
│   └── .env.example
├── frontend/
│   ├── src/
│   │   ├── App.jsx       # État global + appels API
│   │   ├── main.jsx      # Point d'entrée React
│   │   ├── index.css
│   │   └── components/
│   │       ├── Sidebar.jsx # Conversations + upload documents
│   │       ├── ChatArea.jsx # Zone de chat + input
│   │       └── Message.jsx  # Affichage d'un message
│   ├── index.html
│   ├── package.json
│   └── vite.config.js
```

Installation

1. Backend

```
cd backend
python -m venv venv
source venv/bin/activate      # Windows : venv\Scripts\activate

pip install -r requirements.txt

cp .env.example .env
# → remplir MISTRAL_API_KEY, TAVILY_API_KEY, DATABASE_URL dans .env
```

Obtenir les clés API :

- Mistral AI : <https://console.mistral.ai/api-keys> (payant à l'usage, pas de plan gratuit permanent)
- Tavily : <https://tavily.com> (plan gratuit ~1000 recherches/mois)
- PostgreSQL : Supabase (<https://supabase.com>, gratuit) ou une instance locale

```
uvicorn main:app --reload --port 8000
```

Au premier lancement, le dossier `./documents` est créé automatiquement. Place tes fichiers `.pdf`, `.txt` ou `.csv` dedans (ou utilise le bouton upload dans l'interface) pour qu'ils soient indexés.

2. Frontend

```
cd frontend
npm install
```

```
npm run dev
```

Ouvre <http://localhost:5173> — le backend doit tourner sur <http://localhost:8000>.

Ce qui fonctionne dans cette version

- Agent via `create_agent` (API LangChain à jour, plus `AgentExecutor`)
- Streaming réel token par token (`astream_events`, pas de simulation par découpage de mots)
- RAG fonctionnel : ChromaDB + sentence-transformers, recherche par similarité avec seuil
- Mémoire de conversation persistante par session (PostgreSQL)
- Upload et indexation de documents à chaud
- 3 tools : recherche documentaire, recherche web (Tavily), heure actuelle