**🔷 Phase 1 : Définition du projet (10 tâches)**

C'est la phase cruciale pour cadrer le projet et éviter les erreurs coûteuses plus tard.

1. **Définir les objectifs du chatbot**
   * Support client 24/7 (gestion des tickets, FAQ, assistance)
   * Assistant interne pour employés (recherche documentaire, automatisation des tâches)
   * Chatbot marketing (génération de leads, recommandations produit)
   * Assistant conversationnel intelligent (réponses enrichies avec mémoire contextuelle)
2. **Identifier les utilisateurs cibles**
   * Quels types d’utilisateurs vont interagir avec le chatbot ? (Clients, partenaires, employés)
   * Quel est leur niveau technique ? (Grand public, experts, professionnels)
   * Quel canal est le plus pertinent pour eux ?
3. **Lister les cas d’usage prioritaires**
   * Recenser les principales questions et interactions attendues
   * Prioriser les scénarios critiques (urgences, questions fréquentes, transactions)
   * Définir les réponses souhaitées : simples, guidées, interactives
4. **Choisir les canaux de diffusion**
   * Web (intégration sur site avec widget de chat)
   * WhatsApp, Messenger (messagerie instantanée avec API dédiée)
   * Slack, Microsoft Teams (usage interne en entreprise)
   * Applications mobiles (Flutter, React Native)
5. **Déterminer le ton et la personnalité du chatbot**
   * Formel (sérieux, professionnel, précis)
   * Amical (chaleureux, engageant)
   * Expert (données chiffrées, analyses détaillées)
6. **Lister les intégrations nécessaires**
   * CRM (HubSpot, Salesforce)
   * Base de données clients
   * APIs tierces (paiement, Google Calendar, Notion…)
7. **Sélectionner la technologie backend**
   * FastAPI (performance, simplicité, documentation automatique)
   * Flask (léger, modulaire)
   * Django (puissant mais plus structurant)
   * Node.js (si besoin d’un écosystème JavaScript unifié)
8. **Sélectionner la technologie frontend**
   * React.js (flexibilité, communauté active)
   * Next.js (SEO optimisé, performances)
   * Vue.js (simplicité, rapidité)
9. **Déterminer l’infrastructure d’hébergement**
   * **Backend** : AWS Lambda, Google Cloud Run, Railway (scalabilité)
   * **Frontend** : Vercel, Netlify, Firebase Hosting
   * **Base de données** : PostgreSQL, Firebase Firestore, MongoDB
10. **Estimer les coûts et définir un budget**

* OpenAI API (coût variable en fonction des requêtes)
* Hébergement (serveurs, bases de données, monitoring)
* Maintenance et mises à jour

**🔷 Phase 2 : Configuration de l’environnement de développement (10 tâches)**

L’objectif ici est de mettre en place un environnement stable, sécurisé et bien documenté.

1. **Créer un compte OpenAI et récupérer la clé API**

* Stocker la clé API dans un fichier .env pour éviter les fuites

1. **Installer Python et FastAPI pour le backend**

* Utiliser pyenv et venv pour la gestion des versions Python

1. **Installer React.js / Next.js pour le frontend**

* Choisir Next.js pour SSR si SEO et rapidité sont critiques

1. **Configurer un repository GitHub pour le projet**

* Mettre en place un .gitignore
* Ajouter des branches (dev, main, feature)

1. **Installer un serveur local pour tester l’API**

* Postman ou Insomnia pour tester les endpoints API

1. **Configurer un fichier .env pour stocker les clés API**

* Utiliser dotenv en Python et next.config.js côté frontend

1. **Mettre en place un environnement virtuel pour gérer les dépendances**

* python -m venv env && source env/bin/activate

1. **Installer les librairies nécessaires**

* Backend : openai, fastapi, uvicorn, SQLAlchemy, JWT
* Frontend : axios, tailwindcss, Zustand

1. **Configurer Docker pour une future conteneurisation**

* Écrire un Dockerfile et docker-compose.yml

1. **Écrire un README.md pour documenter le projet**

* Présenter l’architecture et les commandes utiles

## 🔷 ****Phase 3 : Développement du backend (20 tâches)****

L’API backend est le cœur du chatbot, elle gère les interactions avec OpenAI, la base de données et l’authentification.

### 📌 **Exemple d’implémentation d’un endpoint avec FastAPI**

python

CopyEdit

from fastapi import FastAPI, HTTPException

import openai

import os

app = FastAPI()

openai.api\_key = os.getenv("OPENAI\_API\_KEY")

@app.post("/chat")

async def chat(input\_text: str):

response = openai.ChatCompletion.create(

model="gpt-4",

messages=[{"role": "user", "content": input\_text}]

)

return {"response": response["choices"][0]["message"]["content"]}

* **Endpoints** : POST /chat, POST /auth, GET /history
* **Base de données** : PostgreSQL avec SQLAlchemy
* **Sécurité** : JWT pour l’authentification
* **Logs** : Mise en place avec logging et Loguru

## 🔷 ****Phase 4 : Développement du frontend (20 tâches)****

L’objectif est d’avoir une UI fluide, intuitive et rapide.

* **Framework** : React.js avec Zustand pour la gestion d’état
* **UI** : Tailwind CSS pour un design moderne
* **Sécurité** : Stockage sécurisé des tokens

### 📌 **Exemple de requête API dans React.js**

jsx

CopyEdit

import { useState } from "react";

import axios from "axios";

export default function Chat() {

const [message, setMessage] = useState("");

const [response, setResponse] = useState("");

const sendMessage = async () => {

const res = await axios.post("/chat", { input\_text: message });

setResponse(res.data.response);

};

return (

<div>

<input type="text" value={message} onChange={e => setMessage(e.target.value)} />

<button onClick={sendMessage}>Envoyer</button>

<p>Réponse: {response}</p>

</div>

);

}

* **Optimisation** : Lazy loading, cache avec SWR
* **Mobile friendly** : Design responsive avec Flexbox/Grid

**🔷 Phase 5-7 : Intégrations, Sécurité et Déploiement**

* Connexion à WhatsApp et Messenger via Twilio / Meta API
* Ajout de monitoring avec Sentry et LogRocket
* Déploiement automatisé avec GitHub Actions