**🔷 Phase 4 : Développement du Frontend (Détails Complets)**

📌 **Objectif :**  
Construire l'interface utilisateur du chatbot en utilisant **Next.js** pour le **rendering côté serveur (SSR)** et **React.js** pour la gestion dynamique. Cette phase inclut également l'intégration du backend (API) et l’optimisation pour offrir une expérience fluide et réactive pour les utilisateurs.

**🔹 1. Configuration de l’Environnement Frontend**

📌 **Tâches :**

* **Initialiser le projet avec Next.js** : Utilisation de create-next-app pour démarrer un projet avec SSR, ce qui est essentiel pour le SEO et la rapidité de chargement.
* **Installer les dépendances essentielles** : react, react-dom, next, tailwindcss pour la mise en page rapide et axios pour les appels API.
* **Configurer TailwindCSS** : Pour un design réactif et moderne sans besoin d'écrire beaucoup de CSS personnalisé.
  + Installer via npm install tailwindcss postcss autoprefixer et ajouter la configuration nécessaire dans tailwind.config.js.
* **Configurer le gestionnaire d’état** : Installer et configurer **Zustand** pour la gestion de l’état global, utile pour stocker des informations utilisateur ou des états du chatbot.

🔧 **Exemple de création d'un projet Next.js** :

npx create-next-app@latest chatbot-frontend

cd chatbot-frontend

npm install axios tailwindcss zustand

**🔹 2. Structuration du Projet Frontend**

📌 **Tâches :**

* **Organiser la structure des dossiers** pour maintenir la lisibilité et la maintenabilité du code. Par exemple :
  + /components : Composants réutilisables comme boutons, champs de texte, etc.
  + /pages : Pages principales comme l'accueil et le chatbot.
  + /hooks : Pour la logique métier (appels API, gestion des états).
  + /styles : Fichiers CSS personnalisés et configuration de Tailwind.

🔧 **Exemple de structure de dossier :**

/chatbot-frontend

/components

/ChatBot

ChatBox.js

Message.js

/pages

index.js

chatbot.js

/hooks

useChatbot.js

/styles

globals.css

tailwind.config.js

**🔹 3. Développement de l’Interface du Chatbot**

📌 **Tâches :**

* **Créer un composant principal ChatBot** qui gère l'affichage des messages. Ce composant sera responsable de l’envoi et de la réception des messages du chatbot.
* **Créer des composants pour afficher les messages** de l’utilisateur et du chatbot sous forme de bulle de conversation. Utiliser un composant Message pour gérer l’apparence des messages.
* **Ajouter des fonctionnalités d’envoi de message** : Gérer l'envoi des messages via un champ de texte et un bouton "Envoyer".
* **Intégrer les réponses du backend** : Appeler l'API du backend (FastAPI) pour obtenir la réponse générée par le chatbot. Utiliser **axios** pour effectuer ces appels API.
* **Gérer les erreurs d’API** et les scénarios de chargement avec un état loading et des messages d’erreur.
* **Ajouter une fonctionnalité d’historique** : Sauvegarder et afficher l’historique des messages de l’utilisateur et du chatbot pour une expérience continue.

🔧 **Exemple de composant ChatBot avec gestion d’état :**

import { useState } from 'react';

import axios from 'axios';

function ChatBot() {

const [messages, setMessages] = useState([]);

const [input, setInput] = useState('');

const [loading, setLoading] = useState(false);

const handleSendMessage = async () => {

if (!input) return;

setMessages([...messages, { text: input, sender: 'user' }]);

setInput('');

setLoading(true);

try {

const response = await axios.post('/api/chatbot', { user\_input: input });

setMessages([...messages, { text: input, sender: 'user' }, { text: response.data.response, sender: 'bot' }]);

} catch (error) {

console.error("Error communicating with backend", error);

setMessages([...messages, { text: 'Error, please try again!', sender: 'bot' }]);

} finally {

setLoading(false);

}

};

return (

<div className="chatbot">

<div className="messages">

{messages.map((msg, index) => (

<div key={index} className={`message ${msg.sender}`}>

{msg.text}

</div>

))}

</div>

<div className="input">

<input

type="text"

value={input}

onChange={(e) => setInput(e.target.value)}

placeholder="Type a message..."

/>

<button onClick={handleSendMessage} disabled={loading}>Send</button>

</div>

</div>

);

}

export default ChatBot;

**🔹 4. Connexion au Backend et Tests de Communication API**

📌 **Tâches :**

* **Appeler l’API de chat** du backend (FastAPI) pour récupérer des réponses basées sur l’entrée utilisateur.
* **Tester les appels API localement** avec Postman ou Insomnia pour s’assurer que l’intégration avec le backend fonctionne.
* **Configurer un proxy** pour les appels API en développement (next.config.js) pour éviter les problèmes CORS entre le frontend et le backend.
* **Configurer le stockage de l’historique des conversations** dans le backend ou localement pour les utilisateurs réguliers.

🔧 **Exemple de configuration du proxy dans Next.js :**

// next.config.js

module.exports = {

async rewrites() {

return [

{

source: '/api/:path\*',

destination: 'http://localhost:8000/api/:path\*', // Pointage vers le backend FastAPI

},

];

},

};

**🔹 5. Optimisation de l'UI et UX**

📌 **Tâches :**

* **Mise en place du design réactif (responsive)** pour que l’application soit fluide sur mobile et desktop, avec **TailwindCSS**.
* **Améliorer l'expérience utilisateur** avec des animations CSS pour rendre la conversation plus fluide. Par exemple, afficher les messages avec une légère animation de défilement.
* **Ajouter un indicateur de "chargement"** (par exemple, un spinner) pendant que le chatbot génère la réponse.
* **Test d’accessibilité** : Assurer une bonne lisibilité, contraste et interaction au clavier pour rendre l'interface accessible à tous.

🔧 **Exemple de code pour animation CSS de messages :**

@keyframes slideIn {

from {

opacity: 0;

transform: translateY(10px);

}

to {

opacity: 1;

transform: translateY(0);

}

}

.message {

animation: slideIn 0.5s ease-out;

}

**🔹 6. Gestion des Données d'Utilisateurs**

📌 **Tâches :**

* **Authentification des utilisateurs** : Intégrer une authentification JWT pour les utilisateurs qui nécessitent une session (étudiants, employés).
* **Stocker les informations d’utilisateur** (nom, ID) dans l'état global ou dans un contexte sécurisé pour personnaliser les interactions du chatbot.
* **Sauvegarder et récupérer les conversations** dans le localStorage ou une base de données pour garder une trace des interactions.

🔧 **Exemple de sauvegarde dans localStorage :**

// Enregistrer une conversation

localStorage.setItem('chatHistory', JSON.stringify(messages));

// Récupérer la conversation

const savedMessages = JSON.parse(localStorage.getItem('chatHistory'));

**🔹 7. Tests et Débogage**

📌 **Tâches :**

* **Tester la réactivité** du site (desktop, mobile, tablette).
* **Utiliser Jest ou React Testing Library** pour tester les composants React (test unitaire, test d’intégration).
* **Faire des tests utilisateurs** pour valider l'interface.

**📌 Livrables Phase 4**

✔️ **Frontend fonctionnel avec intégration du backend**.  
✔️ **Interface utilisateur réactive, fluide et sécurisée**.  
✔️ **Documentation des composants, tests et guide de déploiement**.  
✔️ **Optimisation SEO** avec SSR (Server-Side Rendering) pour améliorer le référencement du chatbot.