```tsx
'use client';

// Importations des dépendances principales
import React, { useState, useEffect, useRef, useCallback } from "react";
import {
  FaComment,
  FaPaperclip,
  FaTimes,
  FaPaperPlane,
  FaMicrophone,
  FaGlobe
} from "react-icons/fa";
import { toast } from "sonner";
import { CircleUserRound } from "lucide-react";
import { getApiUrl } from "@/app/lib/config";
import { v4 as uuidv4 } from "uuid";
import io, { Socket } from 'socket.io-client';

// Interfaces
interface Message {
  id: string;
  content: string;
  sender: "USER" | "RAG";
  timestamp: Date;
}

interface MainSessionProps {
  activeChatId: string | null;
  setActiveChatId: (id: string | null) => void;
  initialMessages: Message[];
  fetchSettings: () => void;
UserChats: () => void;
}

interface ProfilDropdownProps {
  isProfileOpen: boolean;
  setIsProfileOpen: (value: boolean) => void;
  username: string;
  email: string;
  handleLogout: () => void;
}

// Composant de chargement
const ChatLoading = () => (
  <div className="animate-pulse">
    <div className="w-3 h-3 bg-gray-400 rounded-full inline-block mr-1"></div>
    <div className="w-3 h-3 bg-gray-400 rounded-full inline-block mr-1"></div>
    <div className="w-3 h-3 bg-gray-400 rounded-full inline-block"></div>
  </div>
```

```
);

// Composant ProfilDropdown
const ProfilDropdown = ({
  isProfileOpen,
  setIsProfileOpen,
  username,
  email,
  handleLogout
}: ProfilDropdownProps) => {
  const dropdownRef = useRef<HTMLDivElement>(null);

  useEffect(() => {
    const handleClickOutside = (event: MouseEvent) => {
      if (dropdownRef.current && !dropdownRef.current.contains(event.target as
Node)) {
        setIsProfileOpen(false);
      }
    };

    if (isProfileOpen) {
      document.addEventListener("mousedown", handleClickOutside);
    }

    return () => {
      document.removeEventListener("mousedown", handleClickOutside);
    };
  }, [isProfileOpen, setIsProfileOpen]);

  return (
    isProfileOpen && (
      <div
        ref={dropdownRef}
        className="absolute right-0 mt-2 w-64 bg-white rounded-md shadow-lg z-
50 border border-gray-200"
      >
        <div className="p-4 border-b border-gray-200">
          <h3 className="font-medium text-gray-900">Mon Profil</h3>
        </div>
        <div className="p-4">
          <div className="mb-3">
            <p className="text-sm text-gray-500">Nom d'utilisateur</p>
            <p className="font-medium">{username || "Chargement..."}</p>
          </div>
          <div className="mb-4">
            <p className="text-sm text-gray-500">Email</p>
            <p className="font-medium">{email || "Chargement..."}</p>
          </div>
          <button
```

```
            onClick={handleLogout}
            className="w-full py-2 px-4 bg-red-600 text-white rounded-md
hover:bg-red-700 transition-colors"
          >
            Déconnexion
          </button>
        </div>
      </div>
    )
  );
};

// Composant principal MainSession
const MainSession = ({
  activeChatId,
  setActiveChatId,
  initialMessages,
  fetchUserChats,
}: MainSessionProps) => {
  const [messages, setMessages] = useState<Message[]>(initialMessages);
  const [inputValue, setInputValue] = useState("");
  const [isAiTyping, setIsAiTyping] = useState(false);
  const [isRecording, setIsRecording] = useState(false);
  const [isProfileOpen, setIsProfileOpen] = useState(false);

  const [username, setUsername] = useState("");
  const [email, setEmail] = useState("");
  const socketRef = useRef<Socket | null>(null);

  const companyId = "cmbjjpzr20002m0018cd10zn3";

  const messagesEndRef = useRef<HTMLDivElement>(null);
  const inputRef = useRef<HTMLTextAreaElement>(null);

  const scrollToBottom = useCallback(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: "smooth" });
  }, []);

  useEffect(() => {
    console.log('[DEBUG] État messages mis à jour :', messages);
    scrollToBottom();
  }, [messages, isAiTyping, scrollToBottom]);

  const fetchUser = async () => {
    const token = localStorage.getItem("accessToken");
    const userId = localStorage.getItem("user");

    if (!token || !userId) {
      console.error("Vous devez être connecté pour effectuer cette action.");
```

```
      return;
    }

    try {
      const response = await fetch(
        `${getApiUrl(`/user/connected/${userId}`)}`,
        {
          method: "GET",
          headers: {
            Authorization: `Bearer ${token}`,
          },
        }
      );

      if (!response.ok) throw new Error("Erreur lors de la récupération de
l'utilisateur");

      const user = await response.json();
      setUsername(user.username);
      setEmail(user.email);
    } catch (error) {
      console.error("Erreur lors de la récupération :", error);
      toast.error("Erreur lors de la récupération");
    }
  };

  useEffect(() => {
    fetchUser();
  }, []);

  // Initialisation du WebSocket
  useEffect(() => {
    const token = localStorage.getItem("accessToken");
    socketRef.current = io(getApiUrl(''), {
      auth: { token: `Bearer ${token}` },
    });

    socketRef.current.on('connect', () => {
      console.log('[DEBUG] Connecté au WebSocket:', socketRef.current?.id);
      if (activeChatId) {
        socketRef.current?.emit('joinChat', activeChatId);
        console.log('[DEBUG] Rejoint le chat:', activeChatId);
      }
    });

    socketRef.current.on('messageUpdate', (message: Message) => {
      console.log('[DEBUG] Message WebSocket reçu:', message);
      setMessages((prev) => {
        const updated = [...prev];
```

```
        const index = updated.findIndex((m) => m.id === message.id);
        if (index !== -1) {
          updated[index] = {
            id: message.id,
            content: message.content,
            sender: message.sender,
            timestamp: new Date(message.createdAt || Date.now()),
          };
        } else {
          updated.push({
            id: message.id,
            content: message.content,
            sender: message.sender,
            timestamp: new Date(message.createdAt || Date.now()),
          });
        }
        return updated;
      });
      setIsAiTyping(message.sender === 'RAG' && message.content === '');
    });

    socketRef.current.on('disconnect', () => {
      console.log('[DEBUG] Déconnecté du WebSocket');
    });

    socketRef.current.on('connect_error', (error) => {
      console.error('[ERREUR] Erreur de connexion WebSocket:', error);
      toast.error("Erreur de connexion au WebSocket");
    });

    return () => {
      socketRef.current?.disconnect();
    };
  }, [activeChatId]);

  const handleLogout = () => {
    localStorage.removeItem("accessToken");
    localStorage.removeItem("isAdmin");
    localStorage.removeItem("user");
    window.location.href = "/pages/authentication/login";
  };

  const sendMessageToBackend = async (messageContent: string) => {
    console.log('[DEBUG] Début de sendMessageToBackend - Message:',
messageContent);

    const userId = localStorage.getItem("user");
    const token = localStorage.getItem("accessToken");
    if (!userId || !token) {
```

```
      console.error('[ERREUR] Identifiants manquants dans le localStorage');
      toast.error("Session expirée, veuillez vous reconnecter");
      handleLogout();
      return null;
    }

    const body = JSON.stringify({
      userId,
      question: messageContent,
      companyId,
    });

    console.log('[DEBUG] Configuration requête:', { url: '/chats', method:
"POST", body: JSON.parse(body) });

    try {
      const response = await fetch(getApiUrl('/chats'), {
        method: "POST",
        headers: {
          Authorization: `Bearer ${token}`,
          "Content-Type": "application/json",
        },
        body,
      });

      if (!response.ok) {
        const errorText = await response.text();
        console.error('[ERREUR] Requête échouée:', response.status,
errorText);
        throw new Error(`Erreur HTTP: ${response.status} - ${errorText}`);
      }

      const responseData = await response.json();
      console.log('[DEBUG] Réponse finale:', responseData);

      if (responseData.chatResponse?.id) {
        console.log('[DEBUG] Nouveau chat créé, ID:',
responseData.chatResponse.id);
        setActiveChatId(responseData.chatResponse.id);
        socketRef.current?.emit('joinChat', responseData.chatResponse.id);
        console.log('[DEBUG] Rejoint le chat via WebSocket:',
responseData.chatResponse.id);
      }

      fetchUserChats();
      return responseData.chatResponse;

    } catch (error) {
      console.error('[ERREUR] sendMessageToBackend:', error);
```

```
      toast.error(
        error instanceof Error ? error.message : "Erreur lors de l'envoi du
message"
      );
      return null;
    }
  };

  const fetchChatMessages = async (chatId: string) => {
    const token = localStorage.getItem("accessToken");
    const userId = localStorage.getItem("user");

    if (!token || !userId) {
      console.error("Vous devez être connecté pour effectuer cette action.");
      return;
    }

    try {
      const response = await fetch(`${getApiUrl(`/chats/${chatId}`)}`, {
        method: "GET",
        headers: {
          Authorization: `Bearer ${token}`,
        },
      });

      if (!response.ok) {
        const errorText = await response.text();
        console.error('[ERREUR] fetchChatMessages échoué:', response.status,
errorText);
        throw new Error("Erreur lors de la récupération des messages");
      }

      const chat = await response.json();
      console.log('[DEBUG] Chat récupéré via fetchChatMessages :', chat);
      setMessages(chat.messages.map((msg: any) => ({
        id: msg.id,
        content: msg.content,
        sender: msg.sender,
        timestamp: new Date(msg.createdAt),
      })));
    } catch (error) {
      console.error("Erreur lors de la récupération des messages :", error);
      toast.error("Erreur lors de la récupération des messages");
    }
  };

  useEffect(() => {
    console.log('[DEBUG] activeChatId changé :', activeChatId);
    if (activeChatId) {
```

```
      fetchChatMessages(activeChatId);
      socketRef.current?.emit('joinChat', activeChatId);
      console.log('[DEBUG] Rejoint le chat:', activeChatId);
    } else {
      setMessages([]);
    }
  }, [activeChatId]);

  const handleSendMessage = async () => {
    if (!inputValue.trim()) return;

    const tempId = uuidv4();
    const userMessage: Message = {
      id: tempId,
      content: inputValue.trim(),
      sender: "USER",
      timestamp: new Date(),
    };

    console.log('[DEBUG] Ajout message utilisateur :', userMessage);
    setMessages((prev) => [...prev, userMessage]);
    setInputValue("");
    setIsAiTyping(true);

    try {
      await sendMessageToBackend(inputValue.trim());
    } catch (error) {
      console.error('[ERREUR] handleSendMessage :', error);
      toast.error("Erreur de communication avec le serveur");
    }
  };

  const handleKeyDown = (e: React.KeyboardEvent) => {
    if (e.key === "Enter" && !e.shiftKey) {
      e.preventDefault();
      handleSendMessage();
    }
  };

  const toggleRecording = () => {
    if (isRecording) {
      setIsRecording(false);
      setInputValue((prev) => prev + " [Message vocal transcrit]");
    } else {
      setIsRecording(true);
    }
  };

  return (
```

```
    <div className="flex flex-col h-screen bg-white">
      <div className="p-4 border-b border-gray-200 flex items-center justify-
between">
        <h1 className="righteous text-xl font-semibold text-gray-800">
          Perfect Research
        </h1>

        <div className="relative">
          <button
            onClick={() => setIsProfileOpen(!isProfileOpen)}
            className="p-2 rounded-full hover:bg-gray-100"
          >
            <CircleUserRound className="text-gray-700" size={25} />
          </button>

          <ProfilDropdown
            isProfileOpen={isProfileOpen}
            setIsProfileOpen={setIsProfileOpen}
            username={username}
            email={email}
            handleLogout={handleLogout}
          />
        </div>
      </div>

      <div className="flex-1 overflow-y-auto p-4 md:px-6">
        {messages.length === 0 ? (
          <div className="h-full flex flex-col items-center justify-center">
            <div className="max-w-md text-center space-y-4">
              <div className="mb-4">
                <div className="w-16 h-16 mx-auto bg-blue-100 rounded-full
flex items-center justify-center">
                  <FaComment size={28} className="text-blue-500" />
                </div>
              </div>
              <h2 className="righteous text-2xl font-bold text-gray-800">
                Salut, je suis Perfect Chat
              </h2>
              <p className="text-gray-600">
                Comment puis-je vous aider aujourd'hui ? <br />
                Posez-moi une question sur votre recherche ou mémoire.
              </p>
            </div>
          </div>
        ) : (
          <div className="space-y-6 pt-4 pb-20">
            {messages.map((message) => {
              console.log('[DEBUG] Rendu message :', message);
              return (
```

```
                <div
                  key={message.id}
                  className={`flex ${
                    message.sender === "USER" ? "justify-end" : "justify-
start"
                  }`}
                >
                  <div
                    className={`max-w-[70%] rounded-2xl px-4 py-3 ${
                      message.sender === "USER"
                        ? "bg-blue-400 text-white"
                        : "bg-gray-200 text-gray-800"
                    }`}
                  >
                    <p className="whitespace-pre-wrap">{message.content}</p>
                    <div className="text-xs mt-1 opacity-70 text-right">
                      {new Date(message.timestamp).toLocaleTimeString([], {
                        hour: "2-digit",
                        minute: "2-digit",
                      })}
                    </div>
                  </div>
                </div>
              );
            })}

            {isAiTyping && (
              <div className="flex justify-start">
                <div className="bg-gray-200 text-gray-800 rounded-2xl px-4 py-
3 max-w-[200px]">
                  <div className="flex items-center space-x-2">
                    <ChatLoading />
                  </div>
                </div>
              </div>
            )}

            <div ref={messagesEndRef} />
          </div>
        )}
      </div>

      <div className="p-4 border-t border-gray-200 bg-white">
        <div className="max-w-3xl mx-auto">
          <div className="relative bg-gray-100 rounded-xl border border-gray-
300 shadow-sm p-2">
            <textarea
              ref={inputRef}
              value={inputValue}
```

```
              onChange={(e) => setInputValue(e.target.value)}
              onKeyDown={handleKeyDown}
              className="w-full p-4 pl-12 pr-12 rounded-xl resize-none
focus:outline-none bg-transparent border-0"
              rows={1}
              style={{ minHeight: "60px", maxHeight: "150px" }}
              placeholder="Écrivez votre message ici..."
            />

            <div className="absolute left-2 bottom-2 flex gap-2">
              <button className="p-2 rounded-full bg-gray-200 hover:bg-gray-
300">
                <FaGlobe size={16} />
              </button>
            </div>

            <div className="absolute right-2 bottom-2 flex gap-2">
              <button
                onClick={toggleRecording}
                className={`p-2 rounded-full transition-colors ${
                  isRecording
                    ? "bg-red-500 text-white hover:bg-red-600"
                    : "bg-gray-200 hover:bg-gray-300"
                }`}
              >
                <FaMicrophone
                  size={16}
                  className={isRecording ? "animate-pulse" : ""}
                />
              </button>
              <button
                onClick={handleSendMessage}
                disabled={!inputValue.trim()}
                className={`p-2 rounded-full ${
                  inputValue.trim()
                    ? "bg-blue-400 text-white hover:bg-blue-500"
                    : "bg-gray-200 text-gray-500"
                }`}
              >
                <FaPaperPlane size={16} />
              </button>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};
```

```
export default MainSession;
```