ContentIQ Frontend - Complete React App

Build a complete React application with these exact specifications:

Project Structure

```
src/
├── components/
│   ├── ChatWidget.jsx
│   ├── MessageList.jsx
│   ├── InputBar.jsx
│   ├── ContentCard.jsx
│   └── AnalyticsDashboard.jsx
├── hooks/
│   ├── useChat.js
│   └── useAnalytics.js
├── sdk/
│   └── ContentIQSDK.js
├── styles/
│   └── chat.css
└── App.jsx
```

Dependencies to Install

```
{
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "framer-motion": "^10.16.0",
    "lucide-react": "^0.263.1",
    "recharts": "^2.9.0",
    "axios": "^1.6.0",
    "date-fns": "^2.30.0",
    "react-markdown": "^9.0.0",
    "tailwindcss": "^3.3.0"
  }
}
```

useChat.js Hook

```
import { useState, useEffect, useCallback, useRef } from 'react';
```

```
import axios from 'axios';

const API_URL = import.meta.env.VITE_API_URL || 'http://localhost:8000';

export const useChat = (config = {}) => {
  const [messages, setMessages] = useState([]);
  const [isLoading, setIsLoading] = useState(false);
  const [isTyping, setIsTyping] = useState(false);
  const [sessionId] = useState(() =>
    `session_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`
  );
  const [analytics, setAnalytics] = useState(null);
  const eventSourceRef = useRef(null);

  // Load chat history from localStorage
  useEffect(() => {
    const savedMessages = localStorage.getItem(`chat_${sessionId}`);
    if (savedMessages) {
      setMessages(JSON.parse(savedMessages));
    }
  }, [sessionId]);

  // Save messages to localStorage
  useEffect(() => {
    if (messages.length > 0) {
      localStorage.setItem(`chat_${sessionId}`, JSON.stringify(messages));
    }
  }, [messages, sessionId]);

  const sendMessage = useCallback(async (content) => {
    // Add user message immediately
    const userMessage = {
      id: Date.now(),
      role: 'user',
      content,
      timestamp: new Date().toISOString()
    };
    setMessages(prev => [...prev, userMessage]);
    setIsLoading(true);
    setIsTyping(true);

    // Create assistant message placeholder
    const assistantMessageId = Date.now() + 1;
    const assistantMessage = {
      id: assistantMessageId,
      role: 'assistant',
      content: '',
      timestamp: new Date().toISOString(),
```

```
  isStreaming: true
};
setMessages(prev => [...prev, assistantMessage]);

try {
  // Use EventSource for streaming
  const eventSource = new EventSource(
    `${API_URL}/chat?` + new URLSearchParams({
      message: content,
      session_id: sessionId,
      stream: 'true'
    })
  );
  eventSourceRef.current = eventSource;

  let fullResponse = '';
  eventSource.onmessage = (event) => {
    const data = JSON.parse(event.data);

    if (data.chunk) {
      fullResponse += data.chunk;
      setMessages(prev => prev.map(msg =>
        msg.id === assistantMessageId
          ? { ...msg, content: fullResponse }
          : msg
      ));
    }

    if (data.done) {
      setMessages(prev => prev.map(msg =>
        msg.id === assistantMessageId
          ? { ...msg, isStreaming: false, responseTime: data.response_time_ms }
          : msg
      ));
      setIsTyping(false);
      eventSource.close();
    }

    if (data.notification) {
      console.log('Content gap detected:', data.notification);
    }
  };

  eventSource.onerror = (error) => {
    console.error('EventSource error:', error);
    eventSource.close();
    setIsTyping(false);
    setIsLoading(false);
```

```
      };
    } catch (error) {
      console.error('Error sending message:', error);
      setMessages(prev => prev.map(msg =>
        msg.id === assistantMessageId
          ? { ...msg, content: 'Sorry, I encountered an error. Please try again.', isStreaming: false
}
          : msg
      ));
    } finally {
      setIsLoading(false);
    }
  }, [sessionId]);

  const clearHistory = useCallback(() => {
    setMessages([]);
    localStorage.removeItem(`chat_${sessionId}`);
  }, [sessionId]);

  const exportHistory = useCallback(() => {
    const chatData = {
      sessionId,
      exportDate: new Date().toISOString(),
      messages
    };
    const blob = new Blob([JSON.stringify(chatData, null, 2)], { type: 'application/json' });
    const url = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = url;
    a.download = `chat_history_${sessionId}.json`;
    a.click();
    URL.revokeObjectURL(url);
  }, [messages, sessionId]);

  // Fetch analytics
  useEffect(() => {
    const fetchAnalytics = async () => {
      try {
        const response = await axios.get(`${API_URL}/analytics`);
        setAnalytics(response.data);
      } catch (error) {
        console.error('Error fetching analytics:', error);
      }
    };
    fetchAnalytics();
    const interval = setInterval(fetchAnalytics, 5000);
    return () => clearInterval(interval);
  }, []);
```

```
  return {
    messages,
    sendMessage,
    isLoading,
    isTyping,
    sessionId,
    clearHistory,
    exportHistory,
    analytics
  };
};
```

ChatWidget.jsx Component

```
import React, { useState, useRef, useEffect } from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import {
  MessageCircle, X, Send, Mic, MicOff, Download,
  Trash2, Maximize2, Minimize2, Bot, User, Clock,
  TrendingUp, AlertCircle, Sparkles
} from 'lucide-react';
import { useChat } from '../hooks/useChat';
import MessageList from './MessageList';
import InputBar from './InputBar';
import ContentCard from './ContentCard';

const ChatWidget = ({ config = {} }) => {
  const [isOpen, setIsOpen] = useState(false);
  const [isFullscreen, setIsFullscreen] = useState(false);
  const [showAnalytics, setShowAnalytics] = useState(false);
  const [isRecording, setIsRecording] = useState(false);
  const messagesEndRef = useRef(null);

  const {
    messages,
    sendMessage,
    isLoading,
    isTyping,
    sessionId,
    clearHistory,
    exportHistory,
    analytics
  } = useChat(config);

  // Scroll to bottom on new messages
  useEffect(() => {
```

```
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [messages]);

  // Voice recording setup
  const startRecording = () => {
    if (!('webkitSpeechRecognition' in window)) {
      alert('Speech recognition not supported');
      return;
    }

    const recognition = new window.webkitSpeechRecognition();
    recognition.continuous = false;
    recognition.interimResults = false;
    recognition.lang = 'en-US';

    recognition.onresult = (event) => {
      const transcript = event.results[0][0].transcript;
      sendMessage(transcript);
      setIsRecording(false);
    };

    recognition.onerror = () => {
      setIsRecording(false);
    };

    recognition.start();
    setIsRecording(true);
  };

  return (
    <>
      {/* Floating Button */}
      <AnimatePresence>
        {!isOpen && (
          <motion.button
            initial={{ scale: 0 }}
            animate={{ scale: 1 }}
            exit={{ scale: 0 }}
            whileHover={{ scale: 1.1 }}
            whileTap={{ scale: 0.9 }}
            onClick={() => setIsOpen(true)}
            className="fixed bottom-6 right-6 bg-gradient-to-r from-blue-600 to-purple-600
text-white rounded-full p-4 shadow-2xl hover:shadow-3xl transition-all z-50"
          >
            <MessageCircle size={28} />
            {messages.length > 0 && (
              <span className="absolute -top-2 -right-2 bg-red-500 text-white text-xs
rounded-full w-6 h-6 flex items-center justify-center">
```

```
              {messages.length}
            </span>
          )}
        </motion.button>
      )}
    </AnimatePresence>

    {/* Chat Window */}
    <AnimatePresence>
      {isOpen && (
        <motion.div
          initial={{ opacity: 0, y: 100, scale: 0.8 }}
          animate={{ opacity: 1, y: 0, scale: 1 }}
          exit={{ opacity: 0, y: 100, scale: 0.8 }}
          transition={{ type: 'spring', damping: 25 }}
          className={`fixed ${isFullscreen ? 'inset-4' : 'bottom-6 right-6 w-96 h-[600px]'}
bg-white rounded-2xl shadow-2xl flex flex-col overflow-hidden z-50`}
        >
          {/* Header */}
          <div className="bg-gradient-to-r from-blue-600 to-purple-600 text-white p-4 flex
items-center justify-between">
            <div className="flex items-center gap-3">
              <div className="relative">
                <Bot size={24} />
                <span className="absolute -bottom-1 -right-1 w-3 h-3 bg-green-400
rounded-full animate-pulse" />
              </div>
              <div>
                <h3 className="font-bold text-lg">ContentIQ Assistant</h3>
                <p className="text-xs opacity-90">Powered by Contentstack MCP</p>
              </div>
            </div>
            <div className="flex items-center gap-2">
              {analytics && (
                <button
                  onClick={() => setShowAnalytics(!showAnalytics)}
                  className="p-2 hover:bg-white/20 rounded-lg transition-colors"
                  title="Analytics"
                >
                  <TrendingUp size={18} />
                </button>
              )}
              <button
                onClick={() => setIsFullscreen(!isFullscreen)}
                className="p-2 hover:bg-white/20 rounded-lg transition-colors"
              >
                {isFullscreen ? <Minimize2 size={18} /> : <Maximize2 size={18} />}
              </button>
```

```
            <button
              onClick={() => setIsOpen(false)}
              className="p-2 hover:bg-white/20 rounded-lg transition-colors"
            >
              <X size={18} />
            </button>
          </div>
        </div>

        {/* Analytics Panel */}
        {showAnalytics && analytics && (
          <motion.div
            initial={{ height: 0 }}
            animate={{ height: 'auto' }}
            className="bg-gradient-to-r from-blue-50 to-purple-50 p-4 border-b"
          >
            <div className="grid grid-cols-3 gap-4 text-sm">
              <div className="text-center">
                <p className="text-gray-600">Total Queries</p>
                <p className="font-bold text-lg">{analytics.total_queries}</p>
              </div>
              <div className="text-center">
                <p className="text-gray-600">Avg Response</p>
                <p className="font-bold
text-lg">{Math.round(analytics.average_response_time_ms)}ms</p>
              </div>
              <div className="text-center">
                <p className="text-gray-600">Success Rate</p>
                <p className="font-bold text-lg">{Math.round(analytics.success_rate)}%</p>
              </div>
            </div>
            {analytics.content_gaps?.length > 0 && (
              <div className="mt-3 p-2 bg-yellow-100 rounded-lg">
                <p className="text-xs text-yellow-800 flex items-center gap-1">
                  <AlertCircle size={12} />
                  {analytics.content_gaps.length} content gaps detected
                </p>
              </div>
            )}
          </motion.div>
        )}

        {/* Messages Area */}
        <MessageList
          messages={messages}
          isTyping={isTyping}
          messagesEndRef={messagesEndRef}
        />
```

```
          {/* Input Area */}
          <InputBar
            onSendMessage={sendMessage}
            isLoading={isLoading}
            isRecording={isRecording}
            onStartRecording={startRecording}
            onExportHistory={exportHistory}
            onClearHistory={clearHistory}
          />
        </motion.div>
      )}
    </AnimatePresence>
  </>
 );
};

export default ChatWidget;
```

MessageList.jsx Component

```
import React from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import { Bot, User, Clock, Sparkles } from 'lucide-react';
import ReactMarkdown from 'react-markdown';
import ContentCard from './ContentCard';

const MessageList = ({ messages, isTyping, messagesEndRef }) => {
 return (
  <div className="flex-1 overflow-y-auto p-4 space-y-4 bg-gradient-to-b from-gray-50
to-white">
    {messages.length === 0 && (
      <motion.div
        initial={{ opacity: 0, y: 20 }}
        animate={{ opacity: 1, y: 0 }}
        className="text-center py-12"
      >
        <Bot size={48} className="mx-auto text-gray-300 mb-4" />
        <h3 className="text-lg font-semibold text-gray-700 mb-2">
          Welcome to ContentIQ!
        </h3>
        <p className="text-gray-500 text-sm max-w-xs mx-auto">
          Ask me anything about your content. I can help you find information, identify gaps,
and suggest improvements.
        </p>
        <div className="mt-6 space-y-2">
          <SuggestionChip text="Show me available tours" />
```

```
          <SuggestionChip text="What content is missing?" />
          <SuggestionChip text="Analytics dashboard" />
        </div>
      </motion.div>
    )}

    <AnimatePresence>
      {messages.map((message, index) => (
        <motion.div
          key={message.id}
          initial={{ opacity: 0, x: message.role === 'user' ? 20 : -20 }}
          animate={{ opacity: 1, x: 0 }}
          transition={{ delay: index * 0.05 }}
          className={`flex ${message.role === 'user' ? 'justify-end' : 'justify-start'}`}
        >
          <div className={`flex gap-3 max-w-[80%] ${message.role === 'user' ?
'flex-row-reverse' : ''}`}>
            <div className={`flex-shrink-0 w-8 h-8 rounded-full flex items-center justify-center
${
              message.role === 'user' ? 'bg-blue-600' : 'bg-gradient-to-r from-purple-600
to-blue-600'
            }`}>
              {message.role === 'user' ? <User size={16} className="text-white" /> : <Bot
size={16} className="text-white" />}
            </div>
            <div className={`flex flex-col gap-1 ${message.role === 'user' ? 'items-end' :
'items-start'}`}>
              <div className={`px-4 py-2 rounded-2xl ${
                message.role === 'user'
                  ? 'bg-blue-600 text-white'
                  : 'bg-white border border-gray-200 text-gray-800 shadow-sm'
              }`}>
                {message.isStreaming && (
                  <Sparkles size={12} className="inline-block mr-1 animate-pulse" />
                )}
                <ReactMarkdown className="prose prose-sm max-w-none">
                  {message.content}
                </ReactMarkdown>
              </div>
              {message.responseTime && (
                <span className="text-xs text-gray-400 flex items-center gap-1 px-2">
                  <Clock size={10} />
                  {message.responseTime}ms
                </span>
              )}
              {message.contentReferences && (
                <div className="mt-2 space-y-2">
                  {message.contentReferences.map((ref, idx) => (
```

```
                  <ContentCard key={idx} content={ref} />
                ))}
              </div>
            )}
          </div>
        </div>
      </motion.div>
    ))}
  </AnimatePresence>

  {isTyping && (
    <motion.div
      initial={{ opacity: 0 }}
      animate={{ opacity: 1 }}
      className="flex items-center gap-2 text-gray-500"
    >
      <Bot size={20} />
      <div className="flex gap-1">
        <span className="w-2 h-2 bg-gray-400 rounded-full animate-bounce" style={{
animationDelay: '0ms' }} />
        <span className="w-2 h-2 bg-gray-400 rounded-full animate-bounce" style={{
animationDelay: '150ms' }} />
        <span className="w-2 h-2 bg-gray-400 rounded-full animate-bounce" style={{
animationDelay: '300ms' }} />
      </div>
    </motion.div>
  )}
  <div ref={messagesEndRef} />
 </div>
);
};

const SuggestionChip = ({ text, onClick }) => (
 <button
  onClick={onClick}
  className="inline-block px-4 py-2 bg-white border border-gray-200 rounded-full text-sm
text-gray-600 hover:bg-blue-50 hover:border-blue-300 hover:text-blue-600 transition-all"
 >
  {text}
 </button>
);

export default MessageList;
```

InputBar.jsx Component

```
import React, { useState, useRef } from 'react';
```

```
import { Send, Mic, MicOff, Paperclip, Download, Trash2 } from 'lucide-react';
import { motion } from 'framer-motion';

const InputBar = ({
  onSendMessage,
  isLoading,
  isRecording,
  onStartRecording,
  onExportHistory,
  onClearHistory
}) => {
  const [input, setInput] = useState('');
  const [showActions, setShowActions] = useState(false);
  const inputRef = useRef(null);

  const handleSubmit = (e) => {
    e.preventDefault();
    if (input.trim() && !isLoading) {
      onSendMessage(input.trim());
      setInput('');
    }
  };

  const quickActions = [
    { icon: Download, label: 'Export Chat', action: onExportHistory },
    { icon: Trash2, label: 'Clear History', action: onClearHistory }
  ];

  return (
    <div className="border-t bg-white p-4">
      {/* Quick Actions */}
      {showActions && (
        <motion.div
          initial={{ height: 0, opacity: 0 }}
          animate={{ height: 'auto', opacity: 1 }}
          className="flex gap-2 mb-3 pb-3 border-b"
        >
          {quickActions.map((action, idx) => (
            <button
              key={idx}
              onClick={action.action}
              className="flex items-center gap-2 px-3 py-1.5 text-sm text-gray-600
hover:bg-gray-100 rounded-lg transition-colors"
            >
              <action.icon size={14} />
              {action.label}
            </button>
          ))}
```

```
        </motion.div>
      )}

      {/* Input Form */}
      <form onSubmit={handleSubmit} className="flex gap-2">
        <div className="flex-1 flex items-center bg-gray-100 rounded-full px-4
focus-within:bg-white focus-within:ring-2 focus-within:ring-blue-500 transition-all">
          <input
            ref={inputRef}
            type="text"
            value={input}
            onChange={(e) => setInput(e.target.value)}
            placeholder="Ask about your content..."
            disabled={isLoading}
            className="flex-1 bg-transparent py-3 outline-none text-sm"
          />
          <button
            type="button"
            onClick={() => setShowActions(!showActions)}
            className="p-1.5 hover:bg-gray-200 rounded-full transition-colors"
          >
            <Paperclip size={18} className="text-gray-500" />
          </button>
        </div>

        {/* Voice Button */}
        <button
          type="button"
          onClick={onStartRecording}
          disabled={isLoading}
          className={`p-3 rounded-full transition-all ${
            isRecording
              ? 'bg-red-500 text-white animate-pulse'
              : 'bg-gray-100 text-gray-600 hover:bg-gray-200'
          }`}
        >
          {isRecording ? <MicOff size={20} /> : <Mic size={20} />}
        </button>

        {/* Send Button */}
        <button
          type="submit"
          disabled={isLoading || !input.trim()}
          className="p-3 bg-gradient-to-r from-blue-600 to-purple-600 text-white rounded-full
hover:shadow-lg disabled:opacity-50 disabled:cursor-not-allowed transition-all"
        >
          <Send size={20} />
        </button>
```

```
      </form>
    </div>
  );
};

export default InputBar;
```

App.jsx - Main Application

```
import React from 'react';
import ChatWidget from './components/ChatWidget';
import './styles/chat.css';

function App() {
  return (
    <div className="min-h-screen bg-gradient-to-br from-blue-50 via-white to-purple-50">
      {/* Demo Content */}
      <div className="container mx-auto px-4 py-16">
        <div className="text-center mb-12">
          <h1 className="text-4xl font-bold mb-4 bg-gradient-to-r from-blue-600 to-purple-600
bg-clip-text text-transparent">
            ContentIQ Demo
          </h1>
          <p className="text-gray-600 text-lg">
            AI-powered chat platform with Contentstack integration
          </p>
        </div>

        <div className="grid md:grid-cols-3 gap-6 max-w-4xl mx-auto">
          <div className="bg-white rounded-xl p-6 shadow-lg">
            <h3 className="font-bold text-lg mb-2">Italian Tours</h3>
            <p className="text-gray-600 mb-4">Explore beautiful Italy</p>
            <button
              onClick={() => window.dispatchEvent(new CustomEvent('contentiq:message', {
                detail: { message: 'Tell me about Italian tours' }
              }))}
              className="text-blue-600 font-semibold hover:text-blue-800"
            >
              Ask AI →
            </button>
          </div>

          <div className="bg-white rounded-xl p-6 shadow-lg">
            <h3 className="font-bold text-lg mb-2">Tokyo Adventures</h3>
            <p className="text-gray-600 mb-4">Discover modern Japan</p>
            <button
              onClick={() => window.dispatchEvent(new CustomEvent('contentiq:message', {
```

```
              detail: { message: 'Show me Tokyo tour packages' }
            }))}
            className="text-blue-600 font-semibold hover:text-blue-800"
          >
            Ask AI →
          </button>
        </div>

        <div className="bg-white rounded-xl p-6 shadow-lg">
          <h3 className="font-bold text-lg mb-2">Safari Experience</h3>
          <p className="text-gray-600 mb-4">African wildlife adventure</p>
          <button
            onClick={() => window.dispatchEvent(new CustomEvent('contentiq:message', {
              detail: { message: 'What safari tours are available?' }
            }))}
            className="text-blue-600 font-semibold hover:text-blue-800"
          >
            Ask AI →
          </button>
        </div>
      </div>
    </div>

    {/* ContentIQ Chat Widget */}
    <ChatWidget config={{
      apiUrl: 'http://localhost:8000',
      contentstackApiKey: process.env.REACT_APP_CONTENTSTACK_API_KEY,
      contentstackDeliveryToken:
process.env.REACT_APP_CONTENTSTACK_DELIVERY_TOKEN
    }} />
  </div>
 );
}

export default App;
```

Environment Variables (.env)

```
REACT_APP_API_URL=http://localhost:8000
REACT_APP_CONTENTSTACK_API_KEY=your_contentstack_api_key
REACT_APP_CONTENTSTACK_DELIVERY_TOKEN=your_delivery_token
```

Copy all this code into Lovable, install dependencies, and it should work with your backend immediately!