Chatbot for Import-Export

A PROJECT REPORT

Submitted by

RAJNIKANT HIRAPARA

92200133013

ABHAY PADARIYA

92200133017

BACHELOR OF TECHNOLOGY

in

Information And Communication Technology

Marwadi University, Rajkot

Information and Communication Technology

Abstract

The project titled **"Chatbot for Import-Export"** focuses on designing and implementing a web-based intelligent information system for Harivarsh Import & Export Pvt. Ltd. In today's global trade environment, customers and partners expect instant answers about products, shipping procedures, certifications, and compliance. Traditional manual handling of these inquiries is time-consuming and often inconsistent. This project addresses that gap by combining a modern website with two integrated chatbots to automate responses, improve accuracy, and enhance user experience.

The first chatbot acts as a **general import–export assistant**, answering queries related to international trade, product categories, shipping policies, and regulatory procedures. It is built to handle **multiple languages and voice input**, making it more inclusive for users from diverse regions. This multilingual, voice-enabled support lowers communication barriers and ensures that prospective buyers or partners can obtain reliable information quickly and in a format that suits them.

The second chatbot is **company-specific**, tailored to the operations of Harivarsh Import & Export Pvt. Ltd. It delivers direct information about exported goods, destination countries, certificates provided, business hours, and shipment arrangements. For example, it can respond to questions like "Which countries receive your grains?" or "Do you export farm equipment to Europe?" by retrieving structured answers from the company's own database. This capability brings transparency and professionalism to client interactions and reduces the workload of support staff.

From a technological standpoint, the frontend of the system is built using **ReactJS with TypeScript** for a fast, responsive user interface, while the backend is developed with **Node.js and Express** to provide RESTful APIs. All data is securely stored in **MongoDB Atlas**, and the services are deployed on **Vercel** (frontend) and **Render** (backend) to ensure high availability. The company-specific chatbot leverages a **Python microservice** for natural-language processing, enabling accurate and context-aware replies. Together, these components form a scalable and efficient platform that demonstrates the practical application of AI-driven chatbots in the import–export sector.

Overall, the system improves customer service and shows how import–export businesses can adopt modern technologies to remain competitive. Future enhancements may include additional languages, predictive analytics, or integration with social platforms.

Index

CHAPTER 1

DESCRIPTION

1.1 Project Summary

The **"Chatbot for Import-Export"** project is a web-based platform developed for Harivarsh Import & Export Pvt. Ltd. It integrates a responsive website with two intelligent chatbots. The first chatbot provides general import–export information to users in multiple languages with optional voice input. The second chatbot is company-specific and responds to questions about products, countries served, certifications, shipment policies, and contact information. By automating these interactions, the system reduces manual workload, ensures accurate replies, and improves customer engagement.

1.2 Purpose

The primary purpose of this project is to build an **AI-enabled, self-service information system** for the import–export domain. It allows customers, partners, and stakeholders to quickly obtain answers about trade procedures or company details without waiting for human assistance. The platform enhances transparency, saves time, and demonstrates how chatbots can modernize traditional trade communication.

1.3 Features

- **Responsive Website** for Harivarsh Import & Export Pvt. Ltd.
- **Chatbot 1**: Import–export information; supports multilingual text and voice input.
- **Chatbot 2**: Company-specific Q&A (export destinations, product details, certifications, business hours, trial shipments, etc.).
- **Structured Database** using MongoDB Atlas for storing FAQs and chatbot content.
- **Cloud Deployment**: Frontend on Vercel, backend on Render.
- **Python NLP Service** powering the company chatbot for accurate, context-aware replies.
- **User-friendly Interface** with instant responses to queries.

1.4 Technology

- **Frontend Technology**

- Html
- CSS
- JavaScript
- React.js
- Type script

- **Backend Technology**

- Node.js
- Express.js
- Mongo DB Atlas

1.5 Tools

- VS code
- Git hub

CHAPTER 2

2.1 Flow Chart

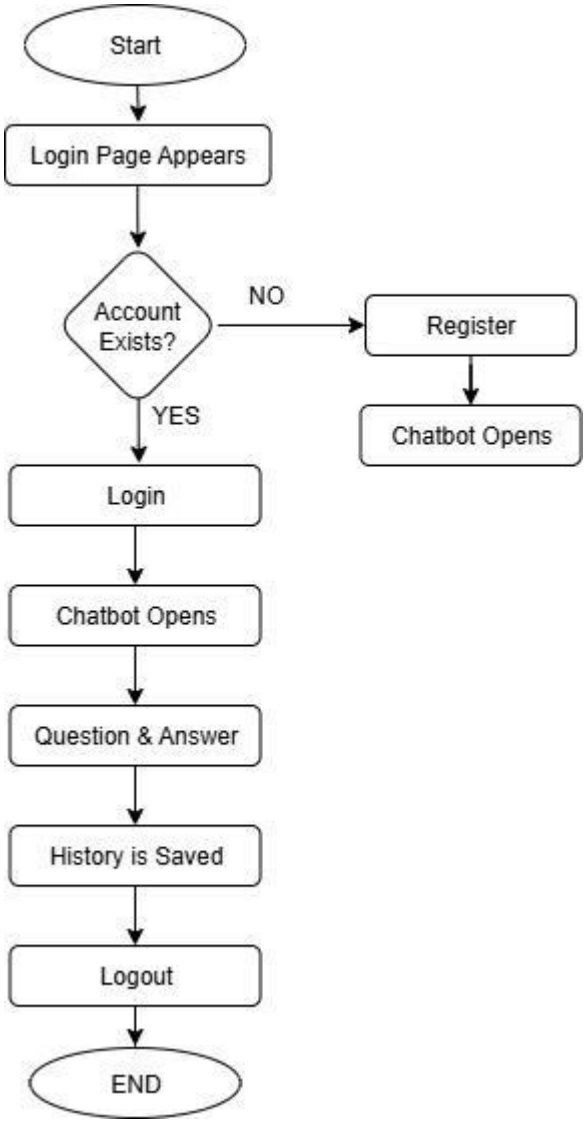


Fig 2.1 Flowchart

CHAPTER 3

CODE SNIPPET & SCREENSHOTS

3.1 Chatbox.tsx

```
import { useState, useEffect, useRef } from "react";
import "../App.css";

// --- Helper function to get auth headers ---
const getAuthHeaders = (): HeadersInit => {
  const userInfoString = localStorage.getItem("userInfo");
  if (!userInfoString) {
    return {};
  }
  const userInfo = JSON.parse(userInfoString);
  return {
    "Content-Type": "application/json",
    "Authorization": `Bearer ${userInfo.token}`,
  };
};

// --- Types ---
interface Message
{ id: number;
  text: string;
  sender: "user" | "ai";
  edited?: boolean;
}

interface Language
  { code: string;
  name: string;
}

interface ConversationMeta
  { conversationId: string;
  lastMessage: string;
  updatedAt: string;
}

interface SpeechRecognition extends EventTarget
  { continuous: boolean;
  interimResults: boolean;
  maxAlternatives: number;
  lang: string;
  start(): void;
  stop(): void;
  onstart: (() => void) | null;
  onresult: ((event: any) => void) | null;
  onend: (() => void) | null;
  onerror: ((event: any) => void) | null;
}
```

```
// --- Supported languages ---
const SUPPORTED_LANGUAGES: Language[] = [
  { code: "en-US", name: "English (US)" },
  { code: "en-IN", name: "English (India)" },
  { code: "hi-IN", name: "हिन् (Hindi)" },
  { code: "gu-IN", name: "ગુજરાતી (Gujarati)" },
  { code: "ta-IN", name: "தமி (Tamil)" },
  { code: "ur-IN", name: "اردو (Urdu)" },
  { code: "bn-IN", name: "বাংলা (Bengali)" },
  { code: "te-IN", name: "తెలు (Telugu)" },
  { code: "mr-IN", name: "मराठी (Marathi)" },
  { code: "es-ES", name: "Español (España)" },
  { code: "fr-FR", name: "Français" },
  { code: "zh-CN", name: "中文 (Mandarin)" },
  { code: "ar-SA", name: "العربية (Arabic)" },
  { code: "ml-IN", name: "മലയ ാളം (Malayalam)" },
  { code: "ne-NE", name: "नेपाल् (Nepali)" },
];

// --- Speech Recognition Setup ---
const SpeechRecognitionAPI =
  (window as any).SpeechRecognition || (window as any).webkitSpeechRecognition;

let recognition: SpeechRecognition | null = null;
if (SpeechRecognitionAPI) {
  recognition = new SpeechRecognitionAPI();
  if (recognition) {
    recognition.continuous = false;
    recognition.interimResults = true;
    recognition.maxAlternatives = 1;
  }
}

// --- Helper: split words with position ---
const splitWordsWithIndex = (text: string) =>
  { const words = text.split(" ");
  let position = 0;
  return words.map((word) =>
    { const start = position;
    position += word.length + 1;
    return { word, start, end: position };
  });
};

// --- Main Chatbot Component ---
export default function Chatbot() {
  // This line determines which backend URL to use for API calls
  const API_BASE = import.meta.env.VITE_API_URL || 'http://localhost:5000';

  const [messages, setMessages] = useState<Message[]>([]);
  const [userInput, setUserInput] = useState("");
  const [isLoading, setIsLoading] = useState(false);
  const [isListening, setIsListening] = useState(false);
```

```
const [selectedLanguage, setSelectedLanguage] =
  useState<string>( SUPPORTED_LANGUAGES[0].code
);
const [voices, setVoices] = useState<SpeechSynthesisVoice[]>([]);
const chatContainerRef = useRef<HTMLDivElement>(null);
const [markdownConverter, setMarkdownConverter] = useState<any>(null);
const [speakingMessageId, setSpeakingMessageId] = useState<number | null>(null);
const [isPaused, setIsPaused] = useState(false);
const [highlightedWordIndex, setHighlightedWordIndex] = useState<number |
  null>( null
);
const [currentWords, setCurrentWords] = useState<
  { word: string; start: number; end: number }[]
>([]);
const currentWordsRef = useRef<{ word: string; start: number; end:
  number }[]>( []
);
const [editingMessageId, setEditingMessageId] = useState<number | null>(null);
const [editingText, setEditingText] = useState("");
const [theme, setTheme] = useState<"light" | "dark">("light");
const [conversationId, setConversationId] = useState<string | null>(null);
const [conversations, setConversations] = useState<ConversationMeta[]>([]);
const [isSidebarOpen, setIsSidebarOpen] = useState(window.innerWidth > 768);

const handleLogout = () =>
  { localStorage.removeItem("userInfo");
  window.location.href = "/login";
};

useEffect(() => {
  const loadVoices = () => setVoices(window.speechSynthesis.getVoices());
  window.speechSynthesis.onvoiceschanged = loadVoices;
  loadVoices();
}, []);

useEffect(() => {
  if ((window as any).showdown)
    { setMarkdownConverter(
      new (window as
        any).showdown.Converter({ tables: true,
        simplifiedAutoLink: true,
        strikethrough: true,
        tasklists: true,
      })
    );
  }
}, []);

useEffect(() => {
  if (chatContainerRef.current)
    { chatContainerRef.current.scrollTop =
    chatContainerRef.current.scrollHeight;
  }
}, [messages]);

const loadConversationsFromDB = async () =>
  { try {
    const headers = getAuthHeaders();
    const res = await fetch(`${API_BASE}/api/conversations`, { headers });
```

```
    if (!res.ok) {
      if (res.status === 401) window.location.href = "/login";
      return;
    }
    const data: ConversationMeta[] = await res.json();
    setConversations(data || []);
    if (!conversationId && data.length > 0)
      { setConversationId(data[0].conversationId);
      await loadChatsFromDB(data[0].conversationId);
    }
  } catch (err) {
    console.error("Error loading conversations:", err);
  }
};

const loadChatsFromDB = async (convId: string) =>
  { try {
    if (!convId) return;
    const headers = getAuthHeaders();
    const res = await fetch(
      `${API_BASE}/api/chats?conversationId=${encodeURIComponent(convId)}`,
      { headers }
    );
     if (!res.ok) {
      if (res.status === 401) window.location.href = "/login";
      return;
    }
    const data = await res.json();
    const mapped: Message[] = data.map((chat: any, i: number) =>
      ({ id: Date.parse(chat.timestamp) + i,
      text: chat.text,
      sender: chat.sender,
    }));
    setMessages(mapped);
  } catch (err) {
    console.error("Error loading chats:", err);
  }
};

const saveChatToDB = async (msg: Message, convId?: string) =>
  { try {
    const cid = convId || conversationId || String(Date.now());
    const headers = getAuthHeaders();

    await fetch(`${API_BASE}/api/chats`,
      { method: "POST",
      headers: headers,  body:
      JSON.stringify({ conv
      ersationId: cid, sender:
      msg.sender, text:
      msg.text,
        timestamp: new Date(),
      }),
    });
    await loadConversationsFromDB();
  } catch (error) {
    console.error("❌ Error saving chat:", error);
  }
};
```

```
const deleteConversation = async (convId: string) =>
  { try {
    const headers = getAuthHeaders();
    await fetch(
      `${API_BASE}/api/conversations/${encodeURIComponent(convId)}`,
      { method: "DELETE", headers }
    );
    await loadConversationsFromDB();
    setConversationId(null);
    const updatedRes = await fetch(`${API_BASE}/api/conversations`, { headers });
    const updated = await updatedRes.json();
    if (updated.length > 0)
      { setConversationId(updated[0].conversationId);
      await loadChatsFromDB(updated[0].conversationId);
    } else
      { setMessages([])
      ;
    }
  } catch (err) {
    console.error("Error deleting conversation:", err);
  }
};

useEffect(() => {
  const userInfo = localStorage.getItem("userInfo");
  if (userInfo) {
    loadConversationsFromDB();
  }
}, []);

const createNewConversation = async () =>
  { const newId = String(Date.now());
  setConversationId(newId);

  const welcomeMessage =
    " Hello! I am the Import/Export AI Assistant. Please select your language and ask me anything about global trade."; const
  welcomeMsg: Message = {
    id: Date.now(),
    text: welcomeMessage,
    sender: "ai",
  };
  setMessages([welcomeMsg]);
  if (window.innerWidth < 768) setIsSidebarOpen(false);
  await saveChatToDB(welcomeMsg, newId);
  await loadConversationsFromDB();
};

const speak = (text: string, langCode: string, messageId: number) =>
  { if (
    !text ||
    typeof window.speechSynthesis === "undefined" ||
    voices.length === 0
  )
    return;

  if (speakingMessageId === messageId) {
    if (isPaused) window.speechSynthesis.resume();
    else window.speechSynthesis.pause();
```

```
setIsPaused(!isPaused);
```

```
    return;
  }

  window.speechSynthesis.cancel();
  const cleanText = text.replace(/(\*|_|`|#|\[.*\]\(.*\))/g, "");
  const utterance = new SpeechSynthesisUtterance(cleanText);

  const words = splitWordsWithIndex(cleanText);
  currentWordsRef.current = words;
  setCurrentWords(words);

  let selectedVoice =
    voices.find((v) => v.lang === langCode) ||
    voices.find((v) => v.lang.startsWith(langCode.split("-")[0]));
  utterance.voice = selectedVoice || null;
  utterance.lang = langCode;

  utterance.onstart = () =>
    { setSpeakingMessageId(messageId);
    setIsPaused(false);
    setHighlightedWordIndex(null);
  };
  utterance.onend = () =>
    { setSpeakingMessageId(null);
    setIsPaused(false);
    setHighlightedWordIndex(null);
    setCurrentWords([]);
    currentWordsRef.current = [];
  };
  utterance.onerror = () =>
    { setSpeakingMessageId(null);
    setIsPaused(false);
    setHighlightedWordIndex(null);
    setCurrentWords([]);
    currentWordsRef.current = [];
  };
  utterance.onboundary = (event: any) => {
    const idx = currentWordsRef.current.findIndex(
      (w) => event.charIndex >= w.start && event.charIndex < w.end
    );
    if (idx !== -1) setHighlightedWordIndex(idx);
  };

  window.speechSynthesis.speak(utterance);
};

const callGeminiAPI = async (prompt: string, history: any[] = []) =>
  { setIsLoading(true);
  const fullHistory = [
    ...history,
    { role: "user", parts: [{ text: prompt }] },
  ];
  const payload = { contents: fullHistory };
  const apiKey = "AIzaSyCVrQzMfM239Hm-gGOklbeWEc_W-pRtuOQ"; // Replace with your key

  try {
    const response = await fetch(
      `https://generativelanguage.googleapis.com/v1beta/models/gemini-1.5-flash:generateContent?key=${apiKey}`,
      {
```

```
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify(payload),
    }
  );
  if (!response.ok) throw new Error(`API Error: ${response.status}`);
  const result = await response.json();
  const aiText =
    result.candidates?.[0]?.content?.parts?.[0]?.text ||
    "Sorry, I couldn't generate a response.";
  const newAiMessage: Message =
    { id: Date.now(),
    text: aiText,
    sender: "ai",
  };
  setMessages((prev) => [...prev, newAiMessage]);
  await saveChatToDB(newAiMessage, conversationId || undefined);
 } catch (error)
  { console.error("Error:",
  error); setMessages((prev)
  => [
    ...prev,
    { id: Date.now(), text: "⚠️ Error connecting to AI.", sender: "ai" },
  ]);
 } finally
  { setIsLoading(false);
 }
};

const handleSendMessage = async (text: string) =>
 { if (!text.trim() || isLoading) return;

 const cid = conversationId || String(Date.now());
 setConversationId(cid);

 if (editingMessageId !== null)
  { setMessages((prev) =>
  prev.map((msg) =>
      msg.id === editingMessageId ? { ...msg, text, edited: true } : msg
    )
  );

  try {
    const headers = getAuthHeaders();
    await fetch(`${API_BASE}/api/chats`,
      { method: "PUT",
      headers: headers,  body:
      JSON.stringify({ conv
      ersationId: cid,
        messageId: editingMessageId,
        text,
      }),
    });
  } catch (err) {
    console.error("❌ Error updating user message:", err);
  }

  const aiMsg = messages.find(
    (msg) => msg.sender === "ai" && msg.id > editingMessageId
```

);

```
    if (aiMsg)
      { try {
        const headers = getAuthHeaders();
        await fetch(`${API_BASE}/api/chats/${aiMsg.id}`,
          { method: "DELETE",
          headers: headers,
        });
      } catch (err) {
        console.error("❌ Error deleting old AI response:", err);
      }
      setMessages((prev) => prev.filter((msg) => msg.id !== aiMsg.id));
    }

    const languageName =
      SUPPORTED_LANGUAGES.find((l) => l.code === selectedLanguage)?.name ||
      "English";
    // const systemPrompt = `You are 'Global Trade AI', an expert assistant specialized ONLY in import and export. IMPORTANT:
If the user asks anything outside import/export or international trade, politely respond: "Sorry, I can only help with import and
export related questions." CRITICAL: Your entire response MUST be in the following language: ${languageName}.`;
const systemPrompt = `
You are 'Global Trade AI', an expert assistant specialized ONLY in international trade, import, and export.

⚠ RULES:
1. Accept questions even if the user makes grammar mistakes, spelling errors, or writes in an informal way.
2. Correct the interpretation of the question internally and always provide a clear, accurate, and professional answer.
3. Answer ONLY topics related to:
  - Import/export processes
  - Customs, duties, tariffs
  - International logistics and shipping
  - Incoterms
  - Trade agreements
  - Export/import documentation (invoice, packing list, bill of lading, etc.)
  - Trade finance, LC, bank guarantees
4.If  the user asks anything unrelated to import/export or international trade, politely respond:
  "Sorry, I can only help with import and export related questions."
5. Always provide answers that are:
  - Concise and to the point
  - Easy to understand
  - Free of unnecessary detail
  - Written in the user's selected language: ${languageName}
6. If a step-by-step explanation is helpful, format it in numbered or bullet points for clarity.
7. Use simple and professional language, even if the question is poorly phrased.

Remember: Your job is to clarify and give **cut-to-cut, correct answers** about import and export.
`;

    const historyForApi = [
      { role: "user", parts: [{ text: systemPrompt }] },
      ...messages
        .filter((msg) => msg.id !== editingMessageId && msg.sender !== "ai")
        .slice(-6)
        .map((msg) => ({
          role: msg.sender === "user" ? "user" : "model",
          parts: [{ text: msg.text }],
        })),
    ];

    setEditingMessageId(null);
    setEditingText("");
```

```
    callGeminiAPI(text, historyForApi);
    return;
  }

  const newUserMessage: Message = { id: Date.now(), text, sender: "user" };
  setMessages((prev) => [...prev, newUserMessage]);
  await saveChatToDB(newUserMessage, cid);

  setUserInput("");

  const languageName =
    SUPPORTED_LANGUAGES.find((l) => l.code === selectedLanguage)?.name ||
    "English";
  const systemPrompt = `You are 'Global Trade AI', an expert assistant specialized ONLY in import and export. IMPORTANT: If the user asks anything outside import/export or international trade, politely respond: "Sorry, I can only help with import and export related questions." CRITICAL: Your entire response MUST be in the following language: ${languageName}.`;

  const historyForApi = [
    { role: "user", parts: [{ text: systemPrompt }] },
    { role: "model", parts: [{ text: "Understood." }] },
    ...messages.slice(-6).map((msg) => ({
      role: msg.sender === "user" ? "user" : "model",
      parts: [{ text: msg.text }],
    })),
  ];

  callGeminiAPI(text, historyForApi);
};

const handleToggleListening = () =>
  { if (!recognition) return;
  if (isListening)
    { recognition.stop();
    setIsListening(false);
    return;
  }

  recognition.lang = selectedLanguage;
  recognition.continuous = false;
  recognition.interimResults = true;
  let finalTranscript = "";

  recognition.onstart = () =>
    { setIsListening(true);
    setUserInput("");
  };
  recognition.onresult = (event: any) =>
    { let interimTranscript = "";
    for (let i = event.resultIndex; i < event.results.length; i++)
      { const transcript = event.results[i][0].transcript;
      if (event.results[i].isFinal) finalTranscript += transcript + " ";
      else interimTranscript += transcript;
    }
    setUserInput((finalTranscript + interimTranscript).trim());
  };
  recognition.onerror = () => setIsListening(false);
  recognition.onend = () => setIsListening(false);
  recognition.start();
};
```

```
const handleSelectConversation = async (convId: string) =>
  { setConversationId(convId);
  if (window.innerWidth < 768) setIsSidebarOpen(false);
  await loadChatsFromDB(convId);
};

return (
  <div className={`background-container ${theme}`}>
    <div className="chat-container">
      <div
        className={`layout ${
          isSidebarOpen ? "sidebar-open" : "sidebar-closed"
        }`}
      >
        <div className="conversations-panel">
          <div className="conversations-header">
            <strong>Conversations</strong>
            <button
              onClick={createNewConversation}
              className="new-convo-btn"
              title="Start a new conversation"
            >
              + New
            </button>
          </div>
          <div className="conversations-list">
            {conversations.length === 0 && (
              <div className="no-convos">No conversations yet — start one!</div>
            )}
            {conversations
              .sort(
                (a, b) =>
                  new Date(b.updatedAt).getTime() -
                  new Date(a.updatedAt).getTime()
              )
              .map((conv) => (
                <div
                  key={conv.conversationId || Math.random()}
                  className={`conversation-item ${
                    conversationId === conv.conversationId ? "active" : ""
                  }`}  onClick={()
                  =>
                    conv.conversationId &&
                    handleSelectConversation(conv.conversationId)
                  }
                >
                  <div className="conv-id">
                    #{conv.conversationId?.slice(-6)}
                  </div>
                  <div className="conv-text">
                    {conv.lastMessage?.slice(0, 80) || "New conversation"}
                  </div>
                  <div className="conv-time">
                    {conv.updatedAt
                      ? new Date(conv.updatedAt).toLocaleString()
                      : ""}
                  </div>
                  <button
```

```
            className="conv-delete"
            onClick={async (e) =>
            { e.stopPropagation();
              if (conv.conversationId)
                await deleteConversation(conv.conversationId);
            }}
           >

           </button>
          </div>
        ))}
      </div>
       <div className="logout-section">
        <button onClick={handleLogout} className="logout-btn">
         Logout
        </button>
       </div>
    </div>
    <div className="chat-main">
     <div className="chat-header">
      <button
       className="menu-toggle"
       onClick={() => setIsSidebarOpen(!isSidebarOpen)}
      >
       ☰
      </button>
      <h1> Import/Export AI Assistant</h1>
      <div>
       <select
        value={selectedLanguage}
        onChange={(e) => setSelectedLanguage(e.target.value)}
        className="language-selector"
       >
        {SUPPORTED_LANGUAGES.map((lang) => (
         <option key={lang.code} value={lang.code}>
          {lang.name}
         </option>
        ))}
       </select>
       <button
        onClick={() => setTheme(theme === "light" ? "dark" : "light")}
        className="theme-toggle"
       >
        {theme === "light" ? "" : "◯"}
       </button>
      </div>
     </div>
     <div className="message-area" ref={chatContainerRef}>
      <div className="message-list">
       {messages.map((msg) => (
        <div
         key={msg.id}
         className={`message-wrapper message-${msg.sender}`}
        >
         {editingMessageId === msg.id ? (
          <div className="edit-message">
           <input
            className="edit-input"
            value={editingText}
```

```
          onChange={(e) => setEditingText(e.target.value)}
        />
        <button
          className="edit-btn"
          onClick={() => {
            setEditingMessageId(null);
            handleSendMessage(editingText);
          }}
        >
          ☑
        </button>
        <button
          className="edit-btn cancel-btn"
          onClick={() => setEditingMessageId(null)}
        >
          ✕
        </button>
      </div>
    ) : (
      <>
        <div className="message-bubble">
          {msg.sender === "ai" &&
          speakingMessageId === msg.id ?
          ( currentWords.map((w, i) => (
              <span  key={i}
                className={
                  i === highlightedWordIndex
                    ? "spoken-word"
                    : ""
                }
              >
                {w.word}{" "}
              </span>
            ))
          ) : (
            <span
              dangerouslySetInnerHTML={{
                __html: markdownConverter
                  ? markdownConverter.makeHtml(msg.text)
                  : msg.text,
              }}
            />
          )}
          {msg.edited && (
            <span className="edited-tag">(edited)</span>
          )}
        </div>
        {msg.sender === "user" && (
          <button
            className="edit-btn"
            onClick={() => {
              setEditingMessageId(msg.id);
              setEditingText(msg.text);
            }}
          >
            🖉
          </button>
        )}
```

```
                {msg.sender === "ai" && (
                 <button
                  onClick={() =>
                   speak(msg.text, selectedLanguage, msg.id)
                  }
                 >
                  {speakingMessageId === msg.id
                   ? isPaused
                     ? "▶️"
                     : "⏸️"
                      : ""}
                 </button>
                )}
               </>
              )}
             </div>
            ))}
            {isLoading && <div className="loading">...</div>}
           </div>
          </div>
          <form  onSubmit={(e)
           =>
           { e.preventDefault();
            handleSendMessage(userInput);
           }}
           className="input-form"
          >
           <button
            type="button"
            onClick={handleToggleListening}
            className={`mic-button
            ${ isListening ? "listening" : ""
            }`}
           >
            {isListening ? "🔘 " : ""}
           </button>
           <input
            value={userInput}
            onChange={(e) => setUserInput(e.target.value)}
            placeholder="Type or speak..."
            className="text-input"
            disabled={isLoading}
           />
           <button
            type="submit"
            className="send-button"
            disabled={!userInput.trim() || isLoading}
           >
            ➤
           </button>
          </form>
         </div>
        </div>
       </div>
      </div>
    );
}
```

3.2 Server.js

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
// --- NEW: Import required packages ---
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
require('dotenv').config(); // Loads variables from .env file

const app = express();
app.use(cors());
app.use(express.json());

// --- MODIFIED: Connect using the secure .env variable ---
mongoose.connect(process.env.MONGO_URI)
  .then(() => console.log("☑ MongoDB connected"))
  .catch((err) => console.error("❌ MongoDB connection error:", err));

// --- NEW: User Schema and Model ---
const userSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true },
});

// Hash password before saving
userSchema.pre('save', async function (next) {
    if (!this.isModified('password')) return next();
    const salt = await bcrypt.genSalt(10);
    this.password = await bcrypt.hash(this.password, salt);
    next();
});

// Method to compare entered password with hashed password
userSchema.methods.matchPassword = async function (enteredPassword) {
    return await bcrypt.compare(enteredPassword, this.password);
};

const User = mongoose.model("User", userSchema);

// --- MODIFIED: Chat Schema now links to a User ---
const chatSchema = new mongoose.Schema({
  user: { // This field links the chat to a specific user
    type: mongoose.Schema.Types.ObjectId,
    ref: 'User',
    required: true
  },
  conversationId: String,
  sender: String,
  text: String,
  timestamp: { type: Date, default: Date.now }
});

const Chat = mongoose.model("Chat", chatSchema);
```

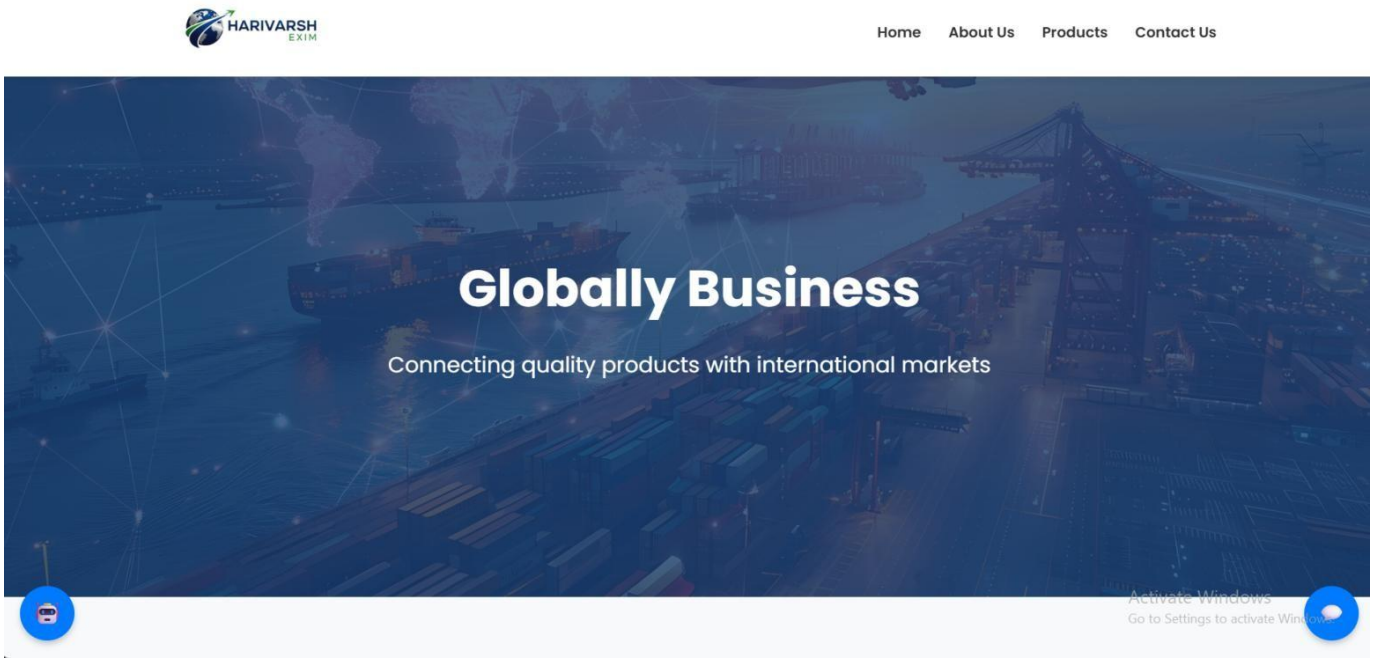
```
// --- NEW: JWT Helper Functions and Middleware ---

// Function to generate a JWT token
const generateToken = (id) => {
  return jwt.sign({ id }, process.env.JWT_SECRET, { expiresIn: '30d' });
};

// Middleware to protect routes
const protect = async (req, res, next) =>
    { let token;
    if (req.headers.authorization && req.headers.authorization.startsWith('Bearer'))
        { try {
            token = req.headers.authorization.split(' ')[1];
            const decoded = jwt.verify(token, process.env.JWT_SECRET);
            // Attach user to the request object
            req.user = await User.findById(decoded.id).select('-password');
            next();
        } catch (error) {
            return res.status(401).json({ message: 'Not authorized, token failed' });
        }
    }
    if (!token) {
        return res.status(401).json({ message: 'Not authorized, no token' });
    }
};

// --- NEW: Authentication Routes ---

// @route   POST /api/auth/register
app.post("/api/auth/register", async (req, res) => {
    const { username, password } = req.body;
    try {
        const userExists = await User.findOne({ username });
        if (userExists) {
            return res.status(400).json({ message: 'User already exists' });
        }
        const user = await User.create({ username, password });
        res.status(201).json({
            _id: user._id,
            username: user.username,
            token: generateToken(user._id)
        });
    } catch (err) {
        res.status(500).json({ error: err.message });
    }
});

// @route   POST /api/auth/login
app.post("/api/auth/login", async (req, res) => {
    const { username, password } = req.body;
    try {
        const user = await User.findOne({ username });
        if (user && (await user.matchPassword(password)))
            { res.json({
                _id: user._id,
                username: user.username,
                token: generateToken(user._id)
            });
        } else {
            res.status(401).json({ message: 'Invalid username or password' });
```

```
    }
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

// --- MODIFIED: All Chat Routes are now protected and user-specific ---

// Save chat (expects conversationId in body)
app.post("/api/chats", protect, async (req, res) => { // Added 'protect' middleware
 try {
  const chatData = {
    ...req.body,
    user: req.user._id // Associate chat with the logged-in user
  };
  const chat = new Chat(chatData);
  await chat.save();
  res.json(chat);
 } catch (err) {
  res.status(500).json({ error: err.message });
 }
});

// Get chats for a conversation
app.get("/api/chats", protect, async (req, res) => { // Added 'protect' middleware
 try {
  const { conversationId } = req.query;
  if (!conversationId) return res.status(400).json({ error: "conversationId required" });
  // Find chats that match conversationId AND the logged-in user
  const chats = await Chat.find({ conversationId, user: req.user._id }).sort({ timestamp: 1 });
  res.json(chats);
 } catch (err) {
  res.status(500).json({ error: err.message });
 }
});

// List conversations (most recent first) for the logged-in user
app.get("/api/conversations", protect, async (req, res) => { // Added 'protect' middleware
try {
  const conversations = await Chat.aggregate([
    { $match: { user: new mongoose.Types.ObjectId(req.user._id) } }, // Only get chats for this user
    { $sort: { conversationId: 1, timestamp: 1 } },
    { $group: {
       _id: "$conversationId",
       lastMessage: { $last: "$text" },
       updatedAt: { $last: "$timestamp" }
      }
    },
    { $sort: { updatedAt: -1 } }
  ]);
  const result = conversations.map(c =>
   ({ conversationId: c._id,
   lastMessage: c.lastMessage,
   updatedAt: c.updatedAt
  }));
  res.json(result);
 } catch (err) {
  res.status(500).json({ error: err.message });
 }
});
```

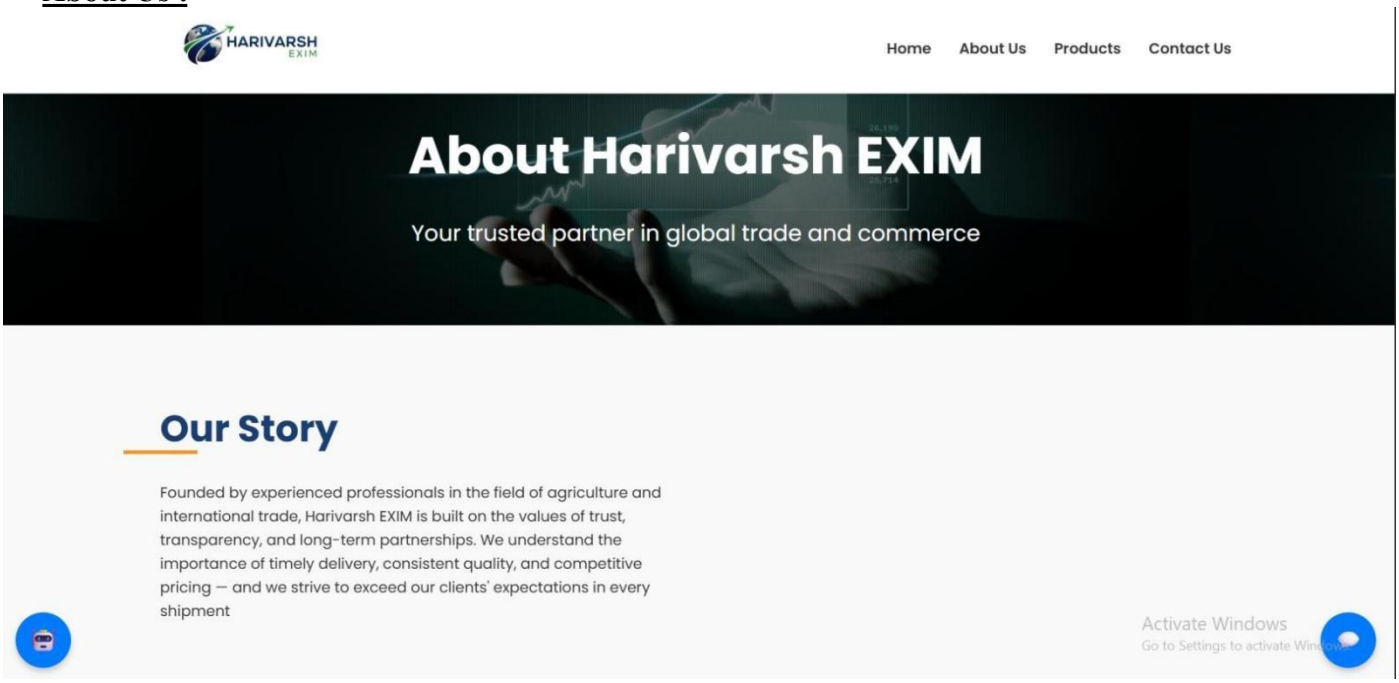
```
// Delete a whole conversation belonging to the logged-in user
app.delete("/api/conversations/:conversationId", protect, async (req, res) => { // Added 'protect' middleware
try {
    const { conversationId } = req.params;
    // Ensure we only delete conversations belonging to this user
    await Chat.deleteMany({ conversationId, user: req.user._id });
    res.json({ message: "Conversation deleted" });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

const PORT = process.env.PORT || 5000;
// NEW CODE
app.listen(PORT, '0.0.0.0', () => console.log(`☑ Backend running on port ${PORT}`));
```

3.1 Output:

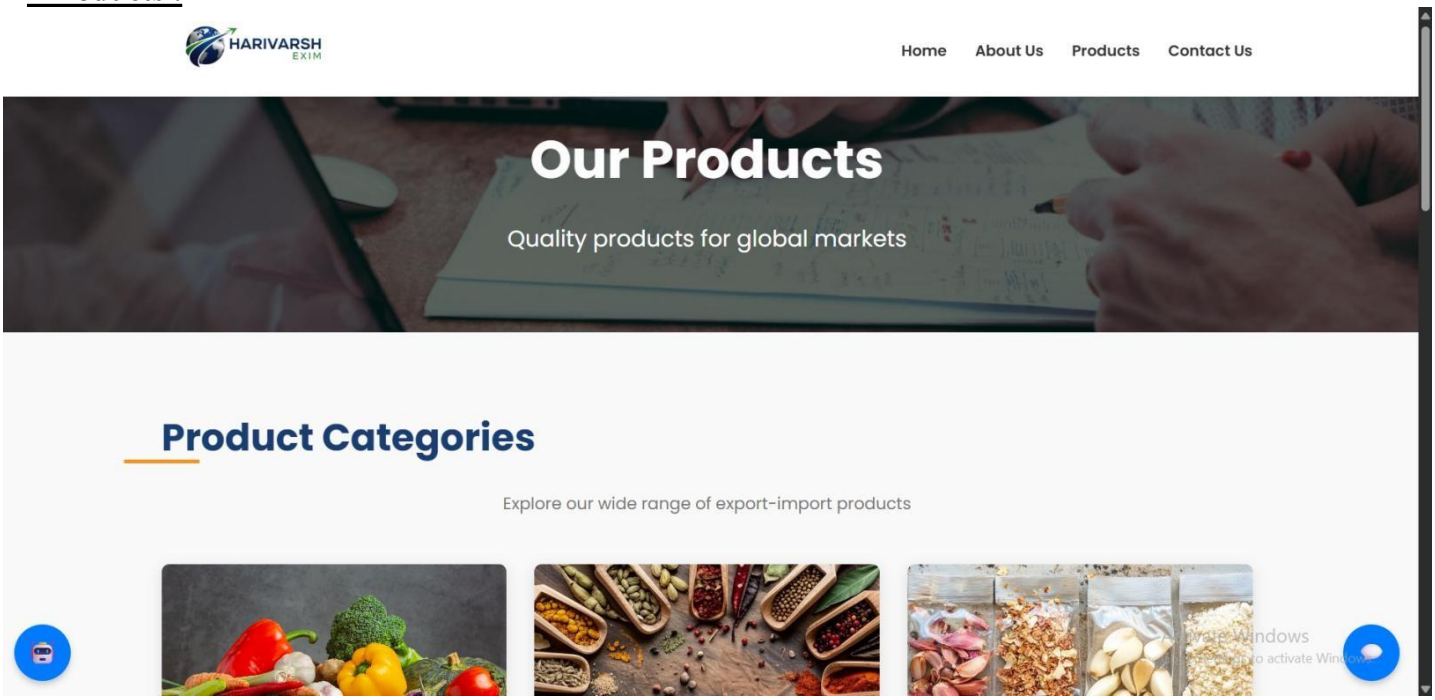
Home page of website :



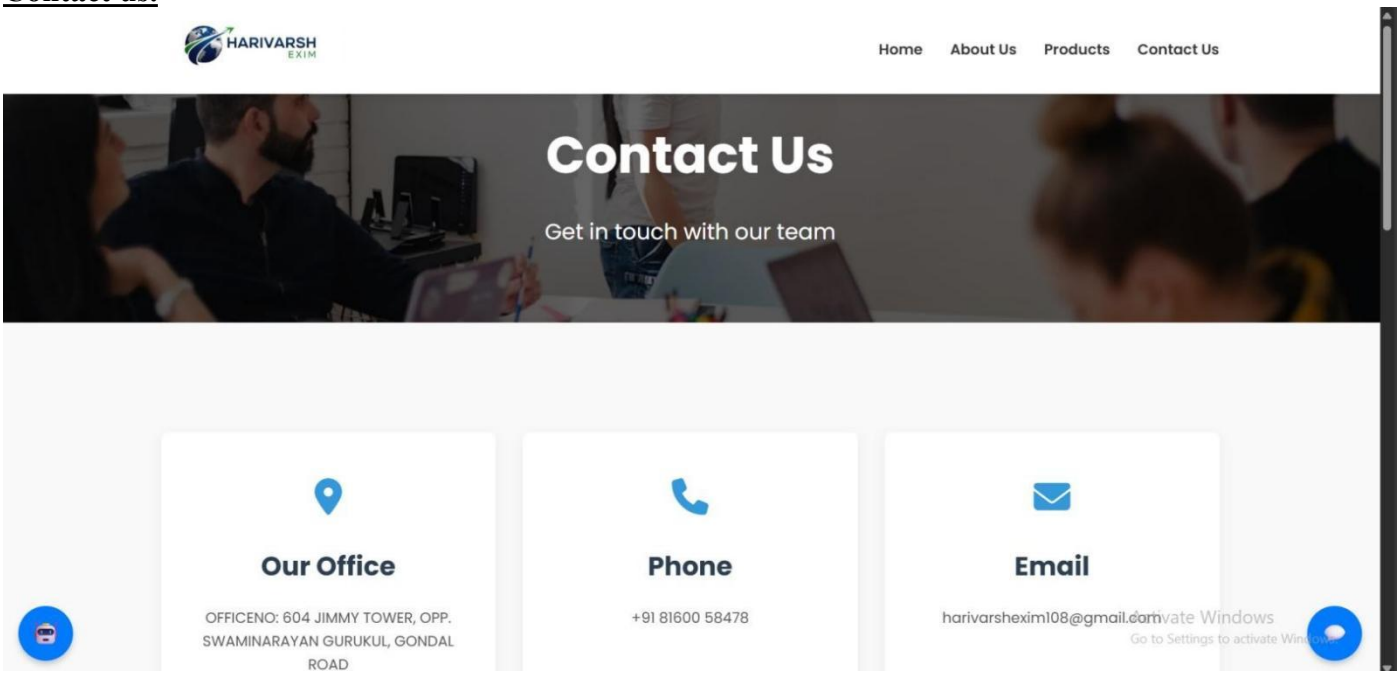
About Us :



Products :

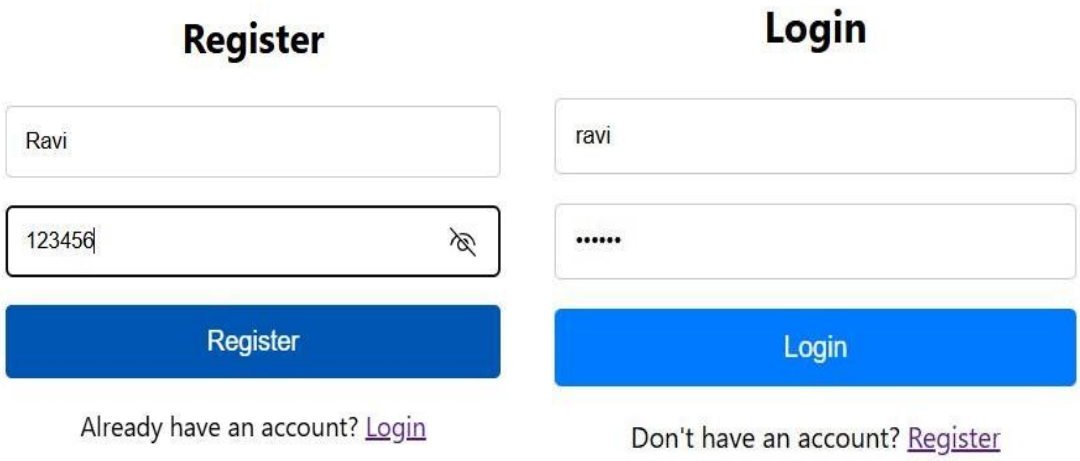


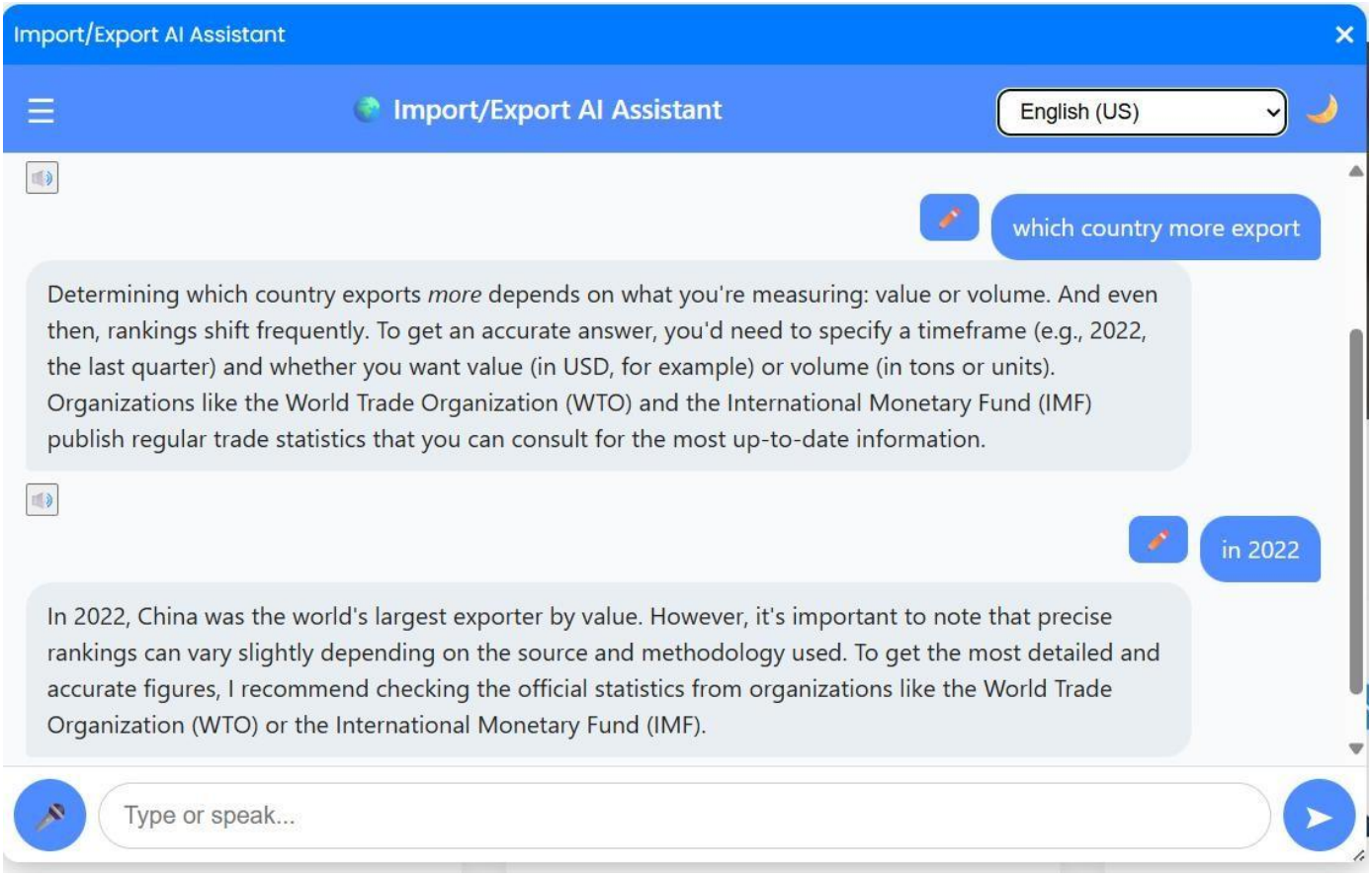
Contact us:



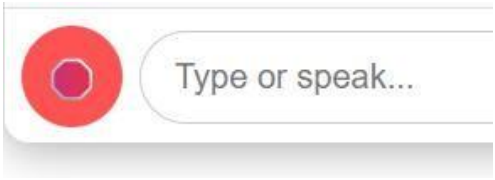
Chatbot 1 : Import Export Related Information

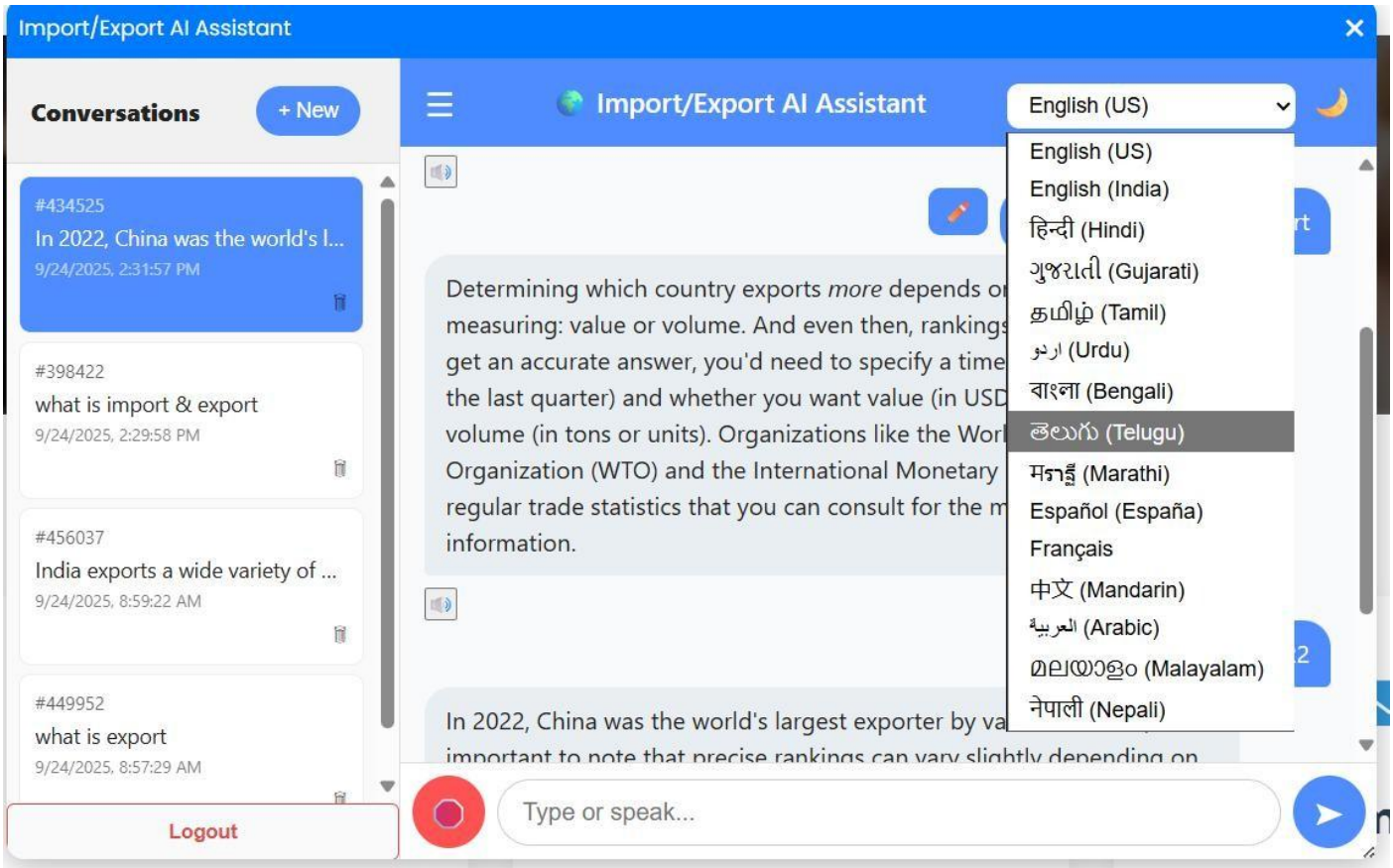
Register & Log in





All lenguages & Voice :





Chatbot 2 : Company Details

3.3 Future Enhancements:

- **Integration with External APIs:**

Connect the chatbot to government or third-party trade portals to automatically fetch real-time data such as updated tariffs, HS codes, and shipping regulations.

- **AI-driven Recommendation Engine:**

Add predictive analytics to suggest potential trade partners, products, or routes based on historical data and market trends.

- **Integration with Social & Messaging Platforms:**

Deploy the chatbot across WhatsApp, Telegram, and Facebook Messenger to widen customer reach.

Innovation and Originality

Introduction

Most chatbots currently available for trade or company support are either very generic (answering only predefined FAQs) or tightly limited to one language and one company. They rarely combine open, multilingual trade information with company-specific knowledge in one seamless system. Our **"Chatbot for Import-Export"** introduces a different approach by blending two specialised chatbots one for general import–export queries and one for Harivarsh Import & Export Pvt. Ltd. into a single, cloud-deployed platform that works in multiple languages and with voice input.

Novel Approach

The innovative aspects of this project are:

- **Dual-chatbot architecture:** Instead of one monolithic bot, the system runs two separate knowledge bases (general import/export + company data) linked under a single interface. This separation improves accuracy, allows independent updates, and makes the system reusable for other companies.
- **Multilingual + voice interaction:** Users can type or speak questions in different languages (English, Hindi, Gujarati, Bengali, etc.) and receive instant answers. Few existing trade chatbots offer real voice-enabled multilingual support.
- **Python NLP microservice + Node.js API:** The company chatbot runs as an independent Python service for natural language processing, while the Node.js backend manages sessions and security. This hybrid design lets each part scale independently.
- **Cloud-native deployment:** Frontend on Vercel, backend and NLP service on Render, database on MongoDB Atlas. This enables 24×7 access without on-premise servers.

Comparison with Existing Solutions

Most import–export information is still provided through static FAQ pages or call centres. Some large logistics firms deploy chatbots, but they are generally:

- **Language-restricted** (usually English only),
- **Closed knowledge bases** (cannot answer outside pre-defined intents),

- **Single-function** (either general trade or company support, not both).

Our approach is **superior** in three ways:

1. **Broader accessibility** – supports many Indian and foreign languages plus voice.
2. **Two-in-one design** – covers both public trade info and private company data under one interface.
3. **Cloud scalability** – the backend and NLP microservice can be scaled or replaced independently, lowering cost and increasing reliability.

Early informal feedback from classmates and faculty showed that the dual-chatbot design is easier to use than juggling separate sites or FAQ lists. This supports our claim of novelty in user experience.

Contribution to the ICT Field

This project demonstrates a practical, low-cost model for deploying AI/NLP chatbots in small or medium-sized trade companies — a sector where most AI solutions are designed for large enterprises. It advances the ICT domain by:

- Showing how **hybrid architectures** (JavaScript frontend + Node API + Python NLP microservice) can be used to build modular AI services.
- Applying **multilingual and voice interfaces** to a real-world business use case, promoting inclusive digital communication.
- Providing a template for future **B2B conversational platforms** that combine public and private knowledge sources securely.

Potential impacts include faster customer response times, lower support costs, and improved transparency in international trade. Future research could add predictive analytics, integration with government trade APIs, or domain adaptation to other industries (logistics, manufacturing, or supply-chain finance).