**https.www.consultancy.africaconsulting-industryit-consulting**

Here’s how to go about creating the **customer support chatbot** using Python:

**1. Project Setup and Design**

**Key Features:**

* **Answering FAQs**: Respond to common questions related to IT consulting services.
* **Service Details**: Provide users with information on the consulting services offered (e.g., IT strategy, ERP services).
* **Live Chat Escalation**: Route to human support if the bot cannot answer.
* **Lead Generation**: Collect user information for follow-up on services.

**Technology Stack:**

* **Python** for backend.
* **Flask** for the API to handle communication.
* **NLP Engine** for query understanding: OpenAI’s GPT-3/4, HuggingFace’s Transformers, or SpaCy.
* **Database** (optional): MongoDB or SQLite to store user queries or feedback.
* **Frontend**: A simple web interface or integration with a platform like **WhatsApp**, **Facebook Messenger**, or a website chat widget.
* **2. Step-by-Step Implementation**
* **Step 1: Set up the Development Environment**
* You’ll need a Python environment. If not already done, create and activate a virtual environment.

# Create virtual environment

python3 -m venv chatbot\_env

source chatbot\_env/bin/activate

# Install necessary libraries

pip install flask openai transformers spacy

python -m spacy download en\_core\_web\_sm # SpaCy model

**Step 2: Create a Simple Flask App**

Flask will serve as the backend API that handles user messages and generates responses.

**Flask App (app.py):**

from flask import Flask, request, jsonify

import openai

app = Flask(\_\_name\_\_)

# OpenAI API key (or use HuggingFace if using transformers)

openai.api\_key = 'your-openai-api-key'

@app.route('/api/chat', methods=['POST'])

def chat():

data = request.json

user\_query = data.get("query")

# OpenAI GPT-3 for generating responses

response = openai.Completion.create(

engine="text-davinci-003",

prompt=user\_query,

max\_tokens=150

)

# Get the response text

response\_text = response.choices[0].text.strip()

return jsonify({"response": response\_text})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

**Step 3: Implement NLP for Query Understanding**

Use a pre-trained model like OpenAI’s GPT to understand customer inquiries related to IT consulting.

You can create a list of common intents and responses such as:

* **Intents**:
  + “What is IT Consulting?”
  + “What are ERP services?”
  + “Tell me about Systems Integration.”
  + “What is the role of an IT Consultant?”

This can also be enhanced by training a custom NLP model or fine-tuning it with more IT-specific data.

**Step 4: Build the Chat Interface**

For a simple web-based interaction, build a chat interface using HTML, CSS, and JavaScript, or integrate this backend into a chatbot frontend like **Chatbot UI**, or **WhatsApp Web**.

Here’s an HTML + JS chat interface example:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>IT Consulting Chatbot</title>

<style>

#chatbox { max-width: 600px; margin: auto; padding: 10px; border: 1px solid #ccc; }

#messages { height: 300px; overflow-y: scroll; border: 1px solid #ccc; padding: 10px; }

.user { color: blue; }

.bot { color: green; }

</style>

</head>

<body>

<div id="chatbox">

<div id="messages"></div>

<input type="text" id="query" placeholder="Ask about IT consulting...">

<button onclick="sendMessage()">Send</button>

</div>

<script>

async function sendMessage() {

const query = document.getElementById('query').value;

const response = await fetch('/api/chat', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ query })

});

const data = await response.json();

document.getElementById('messages').innerHTML += `<div class="user">${query}</div>`;

document.getElementById('messages').innerHTML += `<div class="bot">${data.response}</div>`;

}

</script>

</body>

</html>

**Step 5: Define Sample FAQ Responses**

You can predefine some responses based on typical questions in the IT consulting domain.

Example questions and responses:

* **Question:** "What is IT Consulting?"
  + **Response:** "IT consulting helps businesses use technology to achieve their goals by offering strategic advice, designing IT architectures, and implementing systems like ERP and Data Analytics."
* **Question:** "What services do you offer?"
  + **Response:** "We offer a wide range of services including IT Strategy, ERP services, Systems Integration, IT Security, and more."

For advanced FAQs, you can also make database queries or API calls to dynamically fetch the necessary information.

**Step 6: Integrating with a Database (Optional)**

To log customer queries or save leads, integrate a simple SQLite or MongoDB database.

import sqlite3

# Connect to the database

conn = sqlite3.connect('chatbot.db')

c = conn.cursor()

# Create table

c.execute('''CREATE TABLE IF NOT EXISTS queries (id INTEGER PRIMARY KEY, question TEXT, response TEXT)''')

# Insert user query and response into the database

def log\_query(user\_query, bot\_response):

c.execute("INSERT INTO queries (question, response) VALUES (?, ?)", (user\_query, bot\_response))

conn.commit()

# Example of logging a query

log\_query("What is IT Consulting?", "IT consulting helps businesses...")

**Step 7: Escalation to Human Support (Optional)**

If the bot cannot handle a query, you can route it to human support or prompt the user to contact an agent.

@app.route('/api/chat', methods=['POST'])

def chat():

data = request.json

user\_query = data.get("query")

# Use OpenAI GPT or predefined rules for intent detection

if "escalate" in user\_query.lower():

return jsonify({"response": "I will connect you to a human agent shortly."})

# Otherwise, handle the query with OpenAI or any other NLP logic

response = openai.Completion.create(engine="text-davinci-003", prompt=user\_query, max\_tokens=150)

response\_text = response.choices[0].text.strip()

return jsonify({"response": response\_text})

**3. Testing and Deployment**

**Test Locally:**

Run the Flask app on your local machine.

python app.py

Access the chatbot via http://localhost:5000 in your browser and test the chat interface.

**Deploy to Cloud:**

Once tested locally, you can deploy the chatbot to platforms like **Heroku**, **AWS Lambda**, or **Google Cloud Functions** for production use.

**4. Enhancements**

* **Context Awareness**: Enhance the bot to track ongoing conversations by storing session data in a database (using Redis or simple state management).
* **Multilingual Support**: Use libraries like **Google Translate API** or built-in multilingual capabilities in NLP models to support multiple languages.
* **Analytics**: Add logging and analytics features to monitor chatbot performance, such as query trends and unresolved questions.
* **Integration with CRM**: Integrate the chatbot with CRM platforms like **Salesforce** or **HubSpot** to capture leads generated from customer inquiries.

**Conclusion**

This chatbot will allow you to answer common customer queries related to IT consulting services automatically. By leveraging pre-built models and expanding capabilities with custom logic, you can provide value to your users while offloading routine tasks from your customer service team.