DAY – 31

10 September 2025

To create a **website chatbot** as a **Python developer**, you'll typically follow these steps:

---

1. Define Chatbot Use Case

Decide what your chatbot will do:

- Answer FAQs
- Provide support
- Act as an AI assistant
- Integrate with your website data (RAG)
- etc.

---

2. Choose Chatbot Architecture

You can build your chatbot using:

Approach	Description
Rule-based Bot	Uses if-else logic or regex patterns to reply
ML/NLP-based Bot	Uses models like spaCy, transformers, or Rasa
LLM-based Bot	Uses ChatGPT, Gemini, or open-source LLMs like LLama, Mistral, GPT-Neo
RAG (Retrieval Augmented Generation)	Combines LLM + your website/pdf/DB documents for accurate responses

---

3. Python Libraries for Chatbot

Purpose	Libraries
NLP/ML	spaCy, nltk, transformers, scikit-learn
Chatbot frameworks	Rasa, ChatterBot, LangChain, Haystack, llama-index
LLM API	openai, google-generativeai, transformers (HuggingFace)
Backend/API	Flask, FastAPI, Django
Frontend (Web)	HTML/CSS/JS, React, Chatbot UI, or integration with Botpress, Tidio

---

4. Simple Flask + OpenAI Chatbot Example

Backend (Flask + OpenAI)

```
pip install openai flask flask-cors
# chatbot.py
from flask import Flask, request, jsonify
from flask_cors import CORS
import openai

openai.api_key = "your-openai-api-key"

app = Flask(__name__)
CORS(app)

@app.route("/chat", methods=["POST"])
def chat():
    user_input = request.json.get("message")

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": user_input}]
    )

    return jsonify({"response": response['choices'][0]['message']['content']})
```

```
if __name__ == "__main__":
    app.run(debug=True)
```

Frontend (HTML + JS)

```
<!-- index.html -->
<input id="message" placeholder="Ask something...">
<button onclick="sendMessage()">Send</button>
<div id="chat"></div>

<script>
async function sendMessage() {
    const userMessage = document.getElementById("message").value;
    const res = await fetch("http://127.0.0.1:5000/chat", {
        method: "POST",
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ message: userMessage })
    });
    const data = await res.json();
    document.getElementById("chat").innerHTML += `<p><b>You:</b>
${userMessage}</p><p><b>Bot:</b> ${data.response}</p>`;
}
</script>
```

---

5. Advanced: Create RAG-based Chatbot (PDFs, DBs, Website)

Use LangChain + ChromaDB or FAISS + OpenAI/Gemini to build RAG.

pip install langchain chromadb openai unstructured

Steps:

1. Load your data (PDFs, text, websites)
2. Split and embed it (using OpenAIEmbeddings)

3. Store in vector DB
4. Query + feed context to GPT model

---

6. Open-Source Chatbot Tools

Tool	Use
Rasa	NLP + Dialogue management
LangChain	AI Agent + RAG pipelines
ChatterBot	Easy training for small bots
Botpress	UI + backend chatbot builder
llama-index	Index documents for LLM chat

---

7. Deploy on Website

- Use **Flask/Django** + **Ngrok** or **Render/Vercel**
- Embed the chatbot as a widget using <iframe> or JavaScript

To create a **chatbot for farmers to communicate with website developers** using an **NLP model** that supports **English and Punjabi**, here's a complete plan and sample implementation using Python.

Project Goal:

Build a bilingual chatbot that:

- Accepts **text input in English or Punjabi**.
- Understands farmer queries.
- Replies appropriately (or forwards message to the developer).
- Works on a website frontend using Flask as backend.

Tech Stack:

Feature	Technology
NLP model	transformers (multilingual model)
Translation (optional)	indic-trans, googletrans
Backend	Flask
Frontend	HTML + JS
Deployment (later)	Render / Heroku / Replit

Step-by-Step Implementation

1. Install Required Libraries

pip install flask flask-cors transformers torch

---

2. Code: Flask Backend with Multilingual Support

```
# bilingual_chatbot.py
from flask import Flask, request, jsonify
from flask_cors import CORS
from transformers import pipeline

# Load multilingual model (supports Punjabi + English)
chatbot = pipeline("text-generation", model="mrm8488/GPT2-finetuned-Pytorch")

app = Flask(__name__)
CORS(app)

@app.route("/chat", methods=["POST"])
def chat():
    user_input = request.json.get("message")

    # Generate response (for demo, simple generation)
```

```
    response = chatbot(user_input, max_length=100, num_return_sequences=1)
    message = response[0]["generated_text"]

    return jsonify({"response": message})

if __name__ == "__main__":
    app.run(debug=True)
```

---

3. Frontend (HTML + JavaScript)

```
<!-- chatbot.html -->
<!DOCTYPE html>
<html>
<head>
  <title>Farmer Chatbot</title>
</head>
<body>
  <h2>Chat with Developer (English / Punjabi)</h2>
  <input type="text" id="userInput" placeholder="Type in English or Punjabi">
  <button onclick="sendMessage()">Send</button>
  <div id="chatBox"></div>

  <script>
    async function sendMessage() {
      const message = document.getElementById("userInput").value;

      const response = await fetch("http://127.0.0.1:5000/chat", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ message: message })
      });

      const data = await response.json();
```

```
    document.getElementById("chatBox").innerHTML += `<p><strong>You:</strong> ${message}</p><p><strong>Bot:</strong> ${data.response}</p>`;
  }
 </script>
</body>
</html>
```

Optional: Use Indic-Trans for Punjabi-English Translation

Install:

pip install indic-transliteration

Use for translation if your NLP model is English-only and you want to handle Punjabi via translation.

Multilingual NLP Models You Can Use

Model Name	Description
mrm8488/GPT2-finetuned-Pytorch	GPT2 for multilingual generation
google/mt5-small	mT5 model (translation + generation)
facebook/mbart-large-50-many-to-many-mmt	Strong multilingual model

Future Enhancements

- Add **intent detection** and **responses** using Rasa.
- Add voice support using speechrecognition + gTTS.
- Store chat logs to send to developer.
- WhatsApp/Twilio or email integration.