¡Excelente! Me encanta la iniciativa. Como profesor, te guiaré en la creación de un **Chatbot de Dominio Específico con Inteligencia Conversacional**. Este es un proyecto fantástico que abarca desde el procesamiento del lenguaje natural hasta la arquitectura de software.

**Visión General del Proyecto: "BizChat Assistant"**

Vamos a construir un asistente virtual que:

1. **Comprenda el lenguaje natural** (no solo comandos predefinidos).
2. **Mantenga un diálogo contextual**, recordando lo que se ha hablado en la conversación.
3. **Responda preguntas específicas** de un dominio o negocio que elijas (ej: una tienda online, un consultorio médico, soporte técnico de una app, etc.).
4. **Sea extensible**, para que puedas añadirle nuevas funcionalidades fácilmente en el futuro.

**Características Técnicas y Componentes**

1. **Backend (El Cerebro):** Una API REST en Python usando un framework moderno (FastAPI).

Una **API REST** es un conjunto de reglas y convenciones que permite que aplicaciones distintas se comuniquen de forma estandarizada, simple y eficiente a través de Internet usando el protocolo HTTP.

1. **Procesamiento de Lenguaje Natural (NLP):** Usaremos la biblioteca spaCy para tareas básicas de NLP (análisis léxico, similitud semántica) y scikit-learn para similitud de coseno. Más adelante, podemos integrar modelos de LLM (como GPT con OpenAI API) para hacerlo mucho más potente.
2. **Base de Conocimiento:** Un conjunto de **Preguntas y Respuestas (Q&A)** sobre tu negocio. Empezaremos con algo simple como un archivo JSON. Luego, lo podemos migrar a una base de datos.
3. **Memoria de Conversación:** Implementaremos un sistema simple para recordar el contexto de la conversación reciente.
4. **Frontend (La Cara):** Una interfaz web simple pero funcional usando HTML, CSS y JavaScript para chatear con nuestro backend. Usaremos Fetch API para la comunicación.

**Tecnologías que Aprenderás y Practicarás**

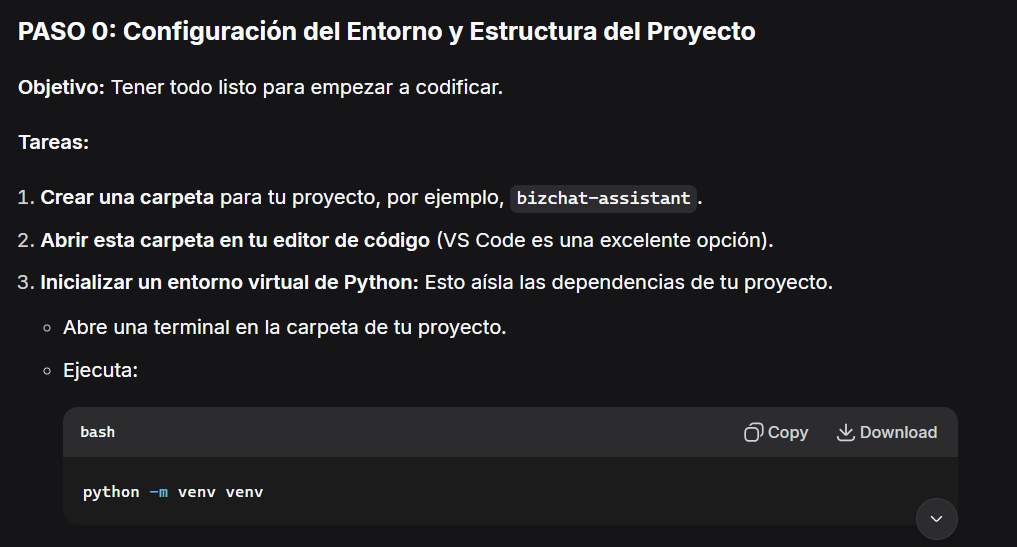
* **Python:** Lógica de programación, manejo de datos.
* **FastAPI:** Creación de APIs REST modernas y rápidas, autodocumentación con Swagger/OpenAPI.
* **POO (Programación Orientada a Objetos):** Diseñaremos clases para el chatbot y la conversación.
* **Procesamiento de Lenguaje Natural (NLP):** Tokenización, stemming, similitud de textos.
* **JavaScript (ES6+):** Manipulación del DOM, eventos, llamadas asíncronas (async/await) a una API.
* **Arquitectura Cliente-Servidor:** Cómo se comunican el frontend y el backend.
* **Manejo de Git:** Para el control de versiones de tu proyecto.

**Plan de Desarrollo por Pasos**

Comenzaremos con una base mínima viable (MVP) y la iremos mejorando iterativamente.

1. **Paso 0:** Configuración del entorno y estructura del proyecto.
2. **Paso 1:** Creación de la base de conocimiento (JSON) y su loader.
3. **Paso 2:** Implementación del núcleo del chatbot (lógica de matching de preguntas).
4. **Paso 3:** Construcción de la API REST con FastAPI.
5. **Paso 4:** Creación del frontend web simple.
6. **Paso 5:** Implementación de la memoria de conversación (contexto).
7. **Paso 6 (Opcional):** Integración con un modelo de LLM (OpenAI) para respuestas más generativas.

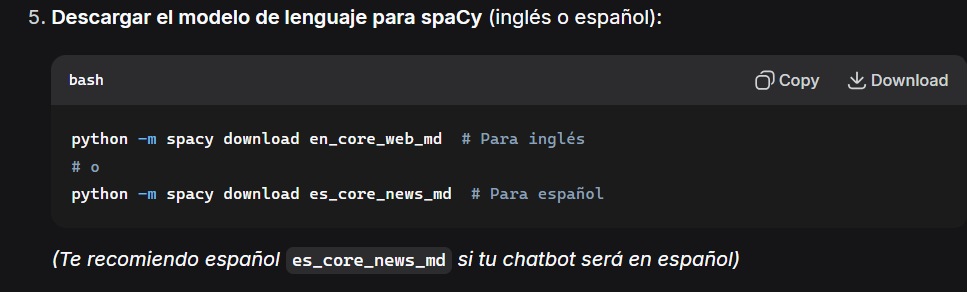
Siempre que completes un paso y lo tengas funcionando, me dices **"Listo para el siguiente paso"** y yo te daré las instrucciones detalladas para el siguiente.



python -m venv venv

1. Actívalo:
   * + **En Windows (PowerShell):** .\venv\Scripts\Activate.ps1
     + **En macOS/Linux:** source venv/bin/activate
   * Verás que el prompt de la terminal cambia a (venv), indicando que el entorno está activo.
2. **Instalar las dependencias iniciales:**

pip install fastapi uvicorn spacy scikit-learn



python -m spacy download en\_core\_web\_md # Para inglés

# o

python -m spacy download es\_core\_news\_md # Para español

**Estructura de archivos iniciales**

bizchat-assistant/

│

├── app/

│ ├── \_\_init\_\_.py

│ ├── main.py # Punto de entrada de FastAPI

│ ├── chatbot.py # Lógica principal del chatbot

│ └── data/

│ └── knowledge\_base.json # Nuestra base de conocimiento

│

├── static/

│ ├── style.css # Estilos CSS

│ └── script.js # Lógica del frontend

│

├── templates/

│ └── index.html # Página principal

│

├── requirements.txt # Dependencias del proyecto

└── venv/ # Tu entorno virtual (NO se sube a Git)

* El **entorno virtual (**venv**)** es crucial en desarrollo Python. Evita conflictos entre versiones de librerías de diferentes proyectos.
* **FastAPI** es el framework que elegimos por su alto rendimiento, facilidad de uso y generación automática de documentación.
* **spaCy** es una librería industrial para NLP. No es la más simple, pero es muy poderosa y te enseñará conceptos reales usados en la industria. El modelo \_md (medium) es un buen balance entre rendimiento y precisión.

PASO 1: La Base de Conocimiento.

Objetivo: Crear el archivo de datos que contendrá toda la información específica de tu negocio o tema, y escribir el código necesario para cargarlo y acceder a él desde Python.

Explicación Conceptual

La base de conocimiento (Knowledge Base) es el "libro de instrucciones" de tu chatbot. Define de lo que puede hablar y qué respuestas debe dar. Para un MVP, la forma más simple es usar un archivo JSON (JavaScript Object Notation), que es un formato ligero y fácil de leer tanto para humanos como para máquinas.

Vamos a estructurarlo como una lista de intenciones. Cada intención representa un tipo de pregunta o acción que el usuario podría quer realizar.

**Tareas a Realizar**

**1. Crear y poblar el archivo**knowledge\_base.json

Navega a la carpeta app/data/ y crea el archivo knowledge\_base.json. Abre el archivo y escribe el siguiente contenido de ejemplo para una tienda online (puedes adaptar los temas y respuestas a tu negocio):

**2. Crear la clase**Chatbot**y el método para cargar la base de conocimiento**

Ahora, ve al archivo app/chatbot.py (que debe estar vacío) y escribe el siguiente código:

**Explicación Docente del Código**

* **Clase**Chatbot**:** Usamos **Programación Orientada a Objetos (POO)**. Una clase es un "molde" que encapsula datos (atributos como intents) y comportamientos (métodos como load\_knowledge\_base). Esto hace el código más organizado, reutilizable y fácil de mantener.
* \_\_init\_\_**:** Este es el *constructor*. Se ejecuta automáticamente cuando creas una nueva instancia de la clase (bot = Chatbot(...)). Su trabajo es inicializar el objeto. Le pasamos la ruta del archivo JSON y inmediatamente llama al método para cargar los datos.
* **Manejo de Errores (**try-except**):** Es crucial manejar posibles errores, como que el archivo no exista o esté corrupto. try-except "intenta" ejecutar un código y "atrapa" (except) cualquier error que pueda ocurrir, permitiéndonos manejarlo elegantemente en lugar de que el programa se cierre de golpe.
* os.path.join**y**\_\_file\_\_**:** Usamos os.path.join para construir rutas de forma que funcionen en Windows, macOS y Linux. \_\_file\_\_ es una variable especial de Python que contiene la ruta del archivo actual (chatbot.py). os.path.dirname(\_\_file\_\_) nos da la carpeta que contiene chatbot.py, y desde ahí navegamos a data/knowledge\_base.json.
* if \_\_name\_\_ == "\_\_main\_\_":**:** Este es un patrón común. El código dentro de este bloque **solo se ejecutará si ejecutas este archivo directamente** (python chatbot.py). Es perfecto para poner código de prueba. Si importas este archivo desde otro (como main.py), este bloque no se ejecutará.

**Prueba del Paso 1**

1. Guarda todos los archivos.
2. En tu terminal, asegúrate de que tu entorno virtual esté activo ((venv) debería aparecer en el prompt).
3. Navega hasta la carpeta app/ (cd app).
4. Ejecuta el comando: python chatbot.py

**Salida esperada:**Base de conocimiento cargada exitosamente. 6 intenciones encontradas.

Tags de intenciones cargadas:

- saludo

- despedida

- horario\_atencion

- metodos\_pago

- agradecimiento

- fallback

**Si ves esta salida, ¡excelente! Has creado con éxito la base de datos de tu chatbot y la has cargado en memoria.**

**PASO 2: El Núcleo de la Inteligencia (Matching de Preguntas).**