

# Ecommerce Copilot Backend - API

---

## Índice

1. Descripción
2. Arquitectura del Sistema
3. Ambiente de Implementación
4. Principales Librerías Utilizadas
5. Guía de Inicio Rápido
6. Arquitectura del Sistema
  1. Diagramas
  2. Actores del Sistema
  3. Casos de Uso
  4. Diagrama Secuencial
  5. Diagrama de Flujo
  6. Diagrama de Arquitectura
7. API Endpoints
8. Estructura de Carpetas y Archivos
9. FAQ's y Problemas Comunes

## Descripción

**Ecommerce Copilot Backend** es una API inteligente basada en FastAPI que proporciona asistencia conversacional y visual para plataformas de comercio electrónico. El sistema utiliza un grafo de conversación con múltiples nodos especializados para ofrecer diferentes tipos de asistencia:

- **Asesoramiento:** Recomendaciones de productos y asistencia general
- **Tooltips Interactivos:** Guías visuales para elementos de la interfaz
- **Guías Paso a Paso:** Secuencias completas de navegación

El sistema integra Google Gemini AI para el procesamiento de lenguaje natural, proporcionando una experiencia de usuario enriquecida y contextual.

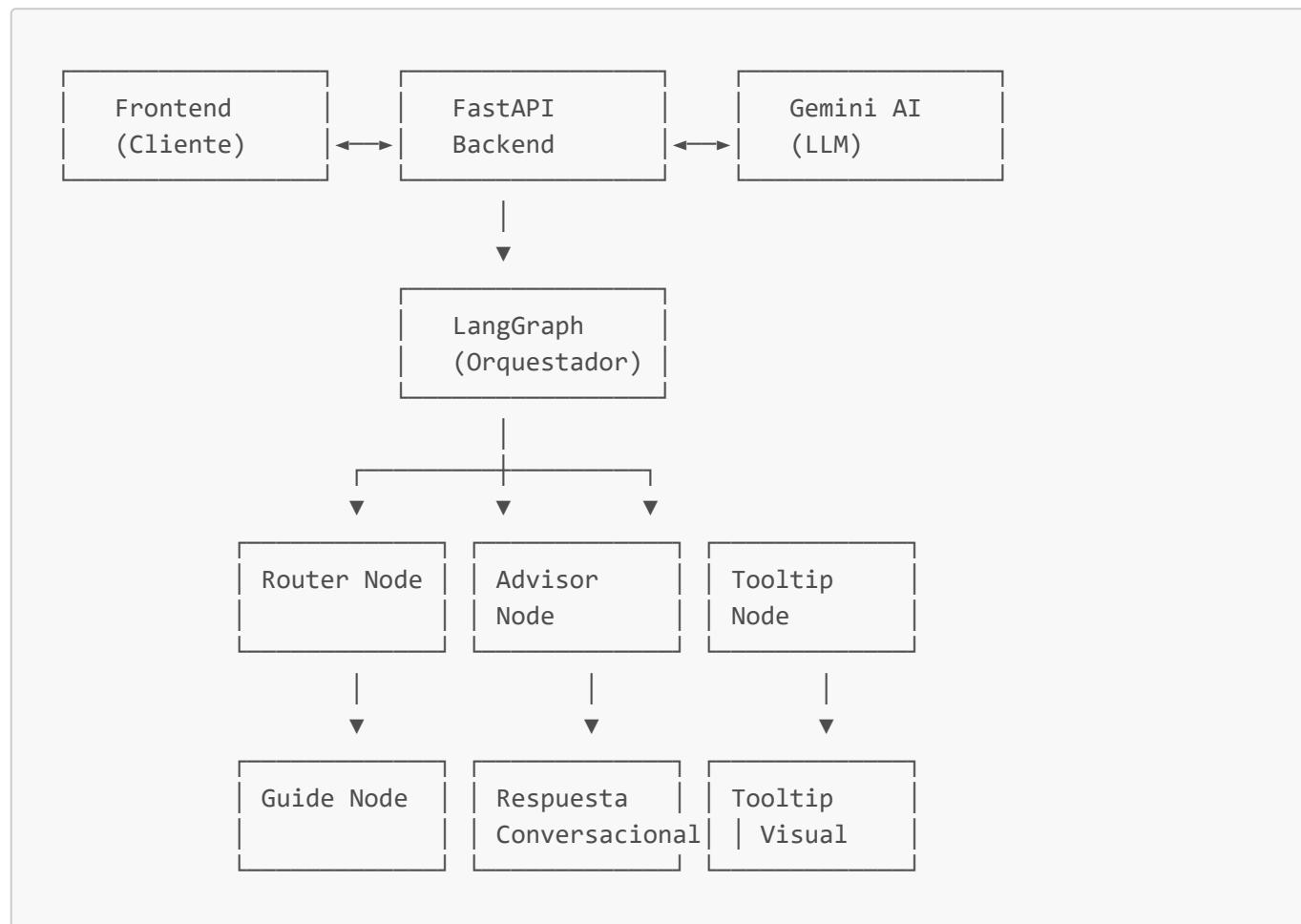
## Características Principales

- ** Asistente Conversacional Inteligente:** Procesamiento de lenguaje natural con clasificación automática de intenciones

- ⓘ **Tooltips Interactivos:** Guías visuales contextuales para elementos de la interfaz
- 📄 **Guías Paso a Paso:** Secuencias completas de navegación para tareas complejas
- 🖼 **Análisis de Imágenes:** Capacidad de procesar y analizar imágenes con Gemini Vision
- 💬 **Grafo de Conversación:** Arquitectura modular con nodos especializados
- ↲ **API REST:** Endpoints optimizados para integración con frontend

## Arquitectura del Sistema

El sistema está construido sobre una arquitectura de grafo de conversación que utiliza LangGraph para orquestar diferentes tipos de asistencia:



## Ambiente de Implementación

Entorno	Ubicación	Descripción
Desarrollo	Local	Entorno de desarrollo local <a href="https://github.com/Nicolas-Pena-Mogollon/ecommerce-copilot-backend">https://github.com/Nicolas-Pena-Mogollon/ecommerce-copilot-backend</a>
Producción	Render	Despliegue automático en Render.com

## Requisitos del Sistema

- **Python:** 3.8+
- **Memoria RAM:** Mínimo 512MB
- **Almacenamiento:** 1GB disponible

- **Red:** Conexión a internet para APIs de Gemini

## Principales Librerías Utilizadas

Librería	Versión	Propósito
<b>fastapi</b>	0.116.1	Framework web para APIs REST
<b>uvicorn</b>	0.35.0	Servidor ASGI para FastAPI
<b>langchain</b>	0.3.27	Framework para aplicaciones LLM
<b>langgraph</b>	0.6.2	Orquestación de grafos de conversación
<b>google-generativeai</b>	0.8.5	Integración con Gemini AI
<b>pydantic</b>	2.11.7	Validación de datos y modelos
<b>python-dotenv</b>	1.1.1	Gestión de variables de entorno

## Guía de Inicio Rápido

### Requisitos Previos

1. **Python 3.8+** instalado
2. **Git** para clonar el repositorio
3. **API Key de Gemini** configurada

### Instalación

```
# 1. Clonar el repositorio
git clone https://github.com/Nicolas-Pena-Mogollon/ecommerce-copilot-backend
cd ecommerce-copilot-backend

# 2. Crear entorno virtual
python -m venv venv
source venv/bin/activate # En Windows: venv\Scripts\activate

# 3. Instalar dependencias
pip install -r requirements.txt

# 4. Configurar variables de entorno
cp .env.example .env
# Editar .env con tu API key de Gemini
```

### Configuración

Crear archivo **.env** con las siguientes variables:

```
GEMINI_API_KEY=tu_api_key_aqui
GEMINI_MODEL=gemini-2.0-flash
```

## Ejecución

```
# Desarrollo (con hot-reload)
uvicorn main:app --reload --host 0.0.0.0 --port 8000

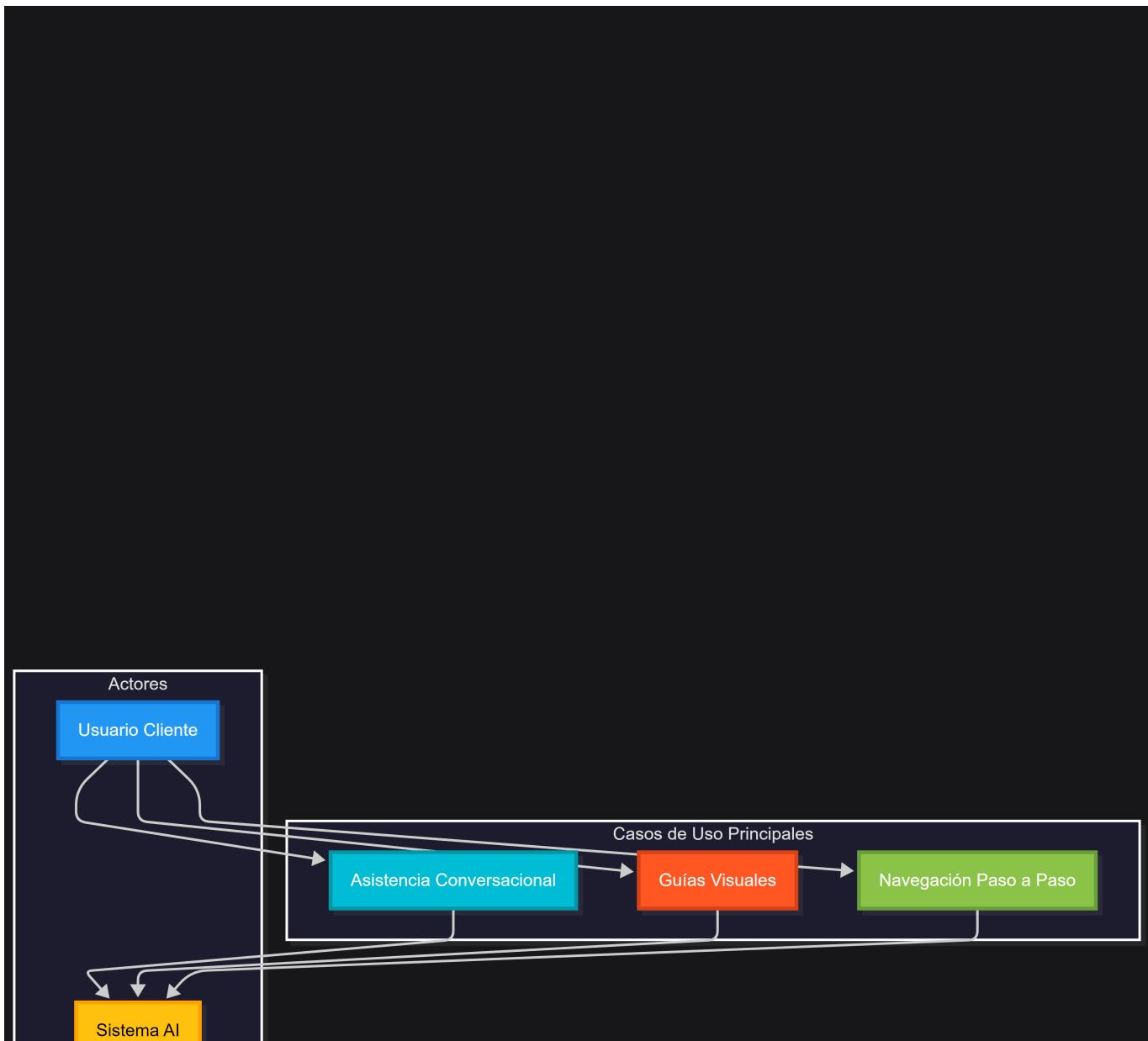
# Producción
uvicorn main:app --host 0.0.0.0 --port 10000
```

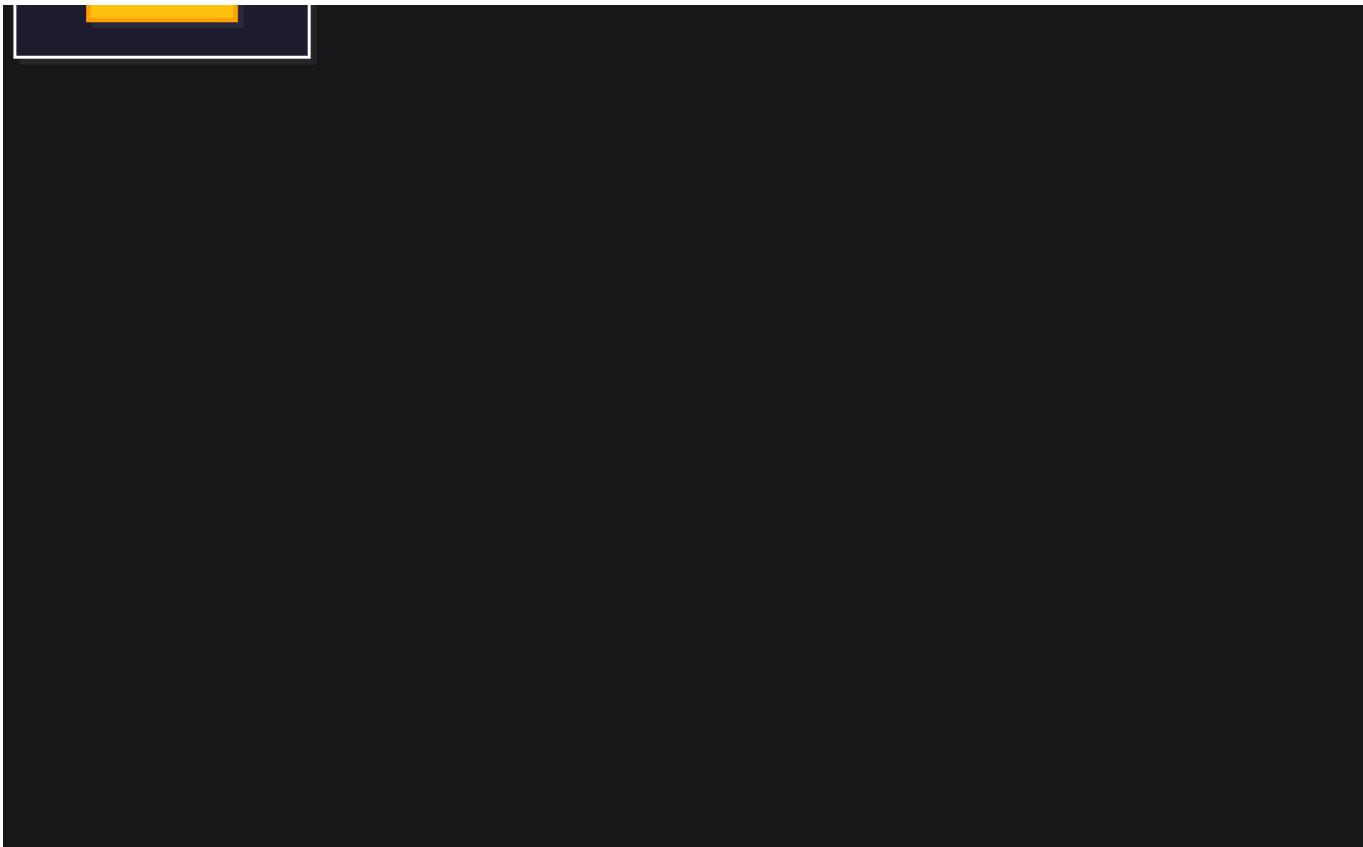
## Verificación

1. **API Docs:** <http://localhost:8000/docs>
2. **Health Check:** <http://localhost:8000/health>

## Diagramas

### Casos de Uso





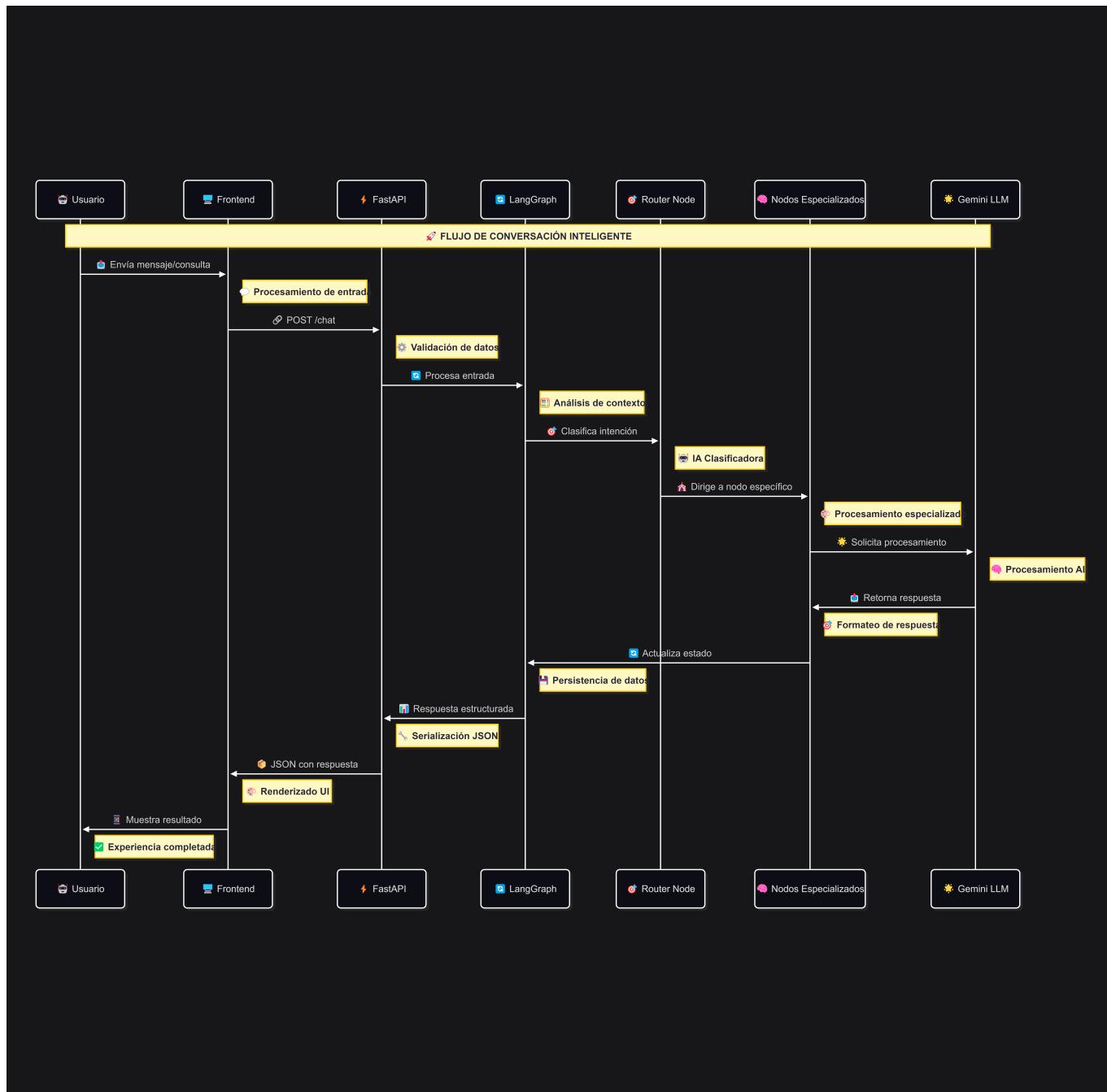
```
graph TD
    subgraph "Casos de Uso Principales"
        A[Asistencia Conversacional]
        B[Guías Visuales]
        C[Navegación Paso a Paso]
    end

    subgraph "Actores"
        E[Usuario Cliente]
        F[Sistema AI]
    end

    E --> A
    E --> B
    E --> C

    A --> F
    B --> F
    C --> F

    style A fill:#00BCD4,stroke:#0097A7,stroke-width:3px,color:#ffffff
    style B fill:#FF5722,stroke:#D84315,stroke-width:3px,color:#ffffff
    style C fill:#8BC34A,stroke:#689F38,stroke-width:3px,color:#ffffff
    style E fill:#2196F3,stroke:#1976D2,stroke-width:3px,color:#ffffff
    style F fill:#FFC107,stroke:#FFA000,stroke-width:3px,color:#000000
```



```

sequenceDiagram
    participant U as 🤖 Usuario
    participant F as 💻 Frontend
    participant A as ⚡ FastAPI
    participant G as 💾 LangGraph
    participant R as ⚙ Router Node
    participant N as 💬 Nodos Especializados
    participant L as 🌟 Gemini LLM

    Note over U,L: 💾 FLUJO DE CONVERSACIÓN INTELIGENTE

```

U->>F: 💬 Envía mensaje/consulta

Note right of F: 💾 Procesamiento de entrada

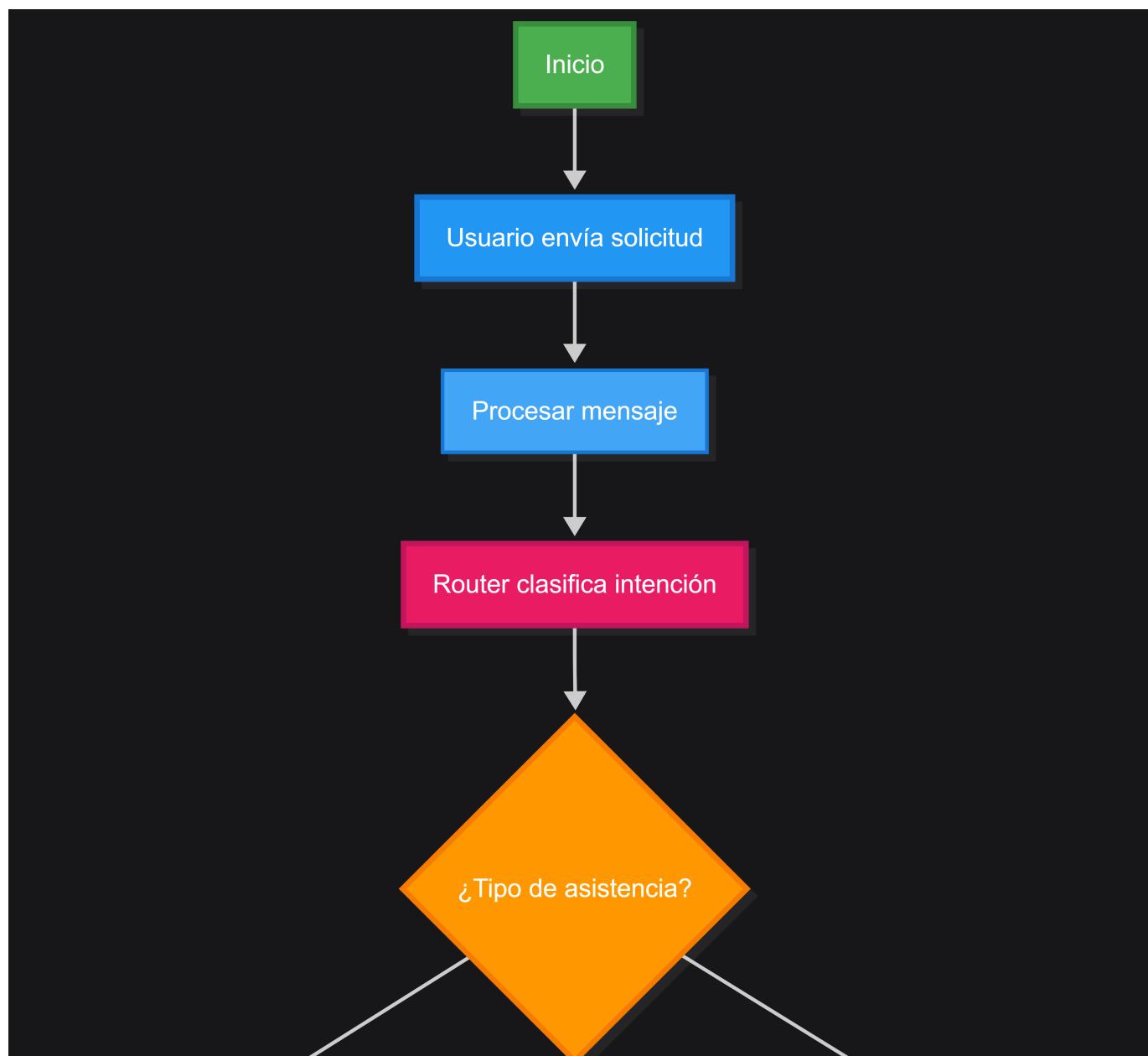
F->>A: 📈 POST /chat

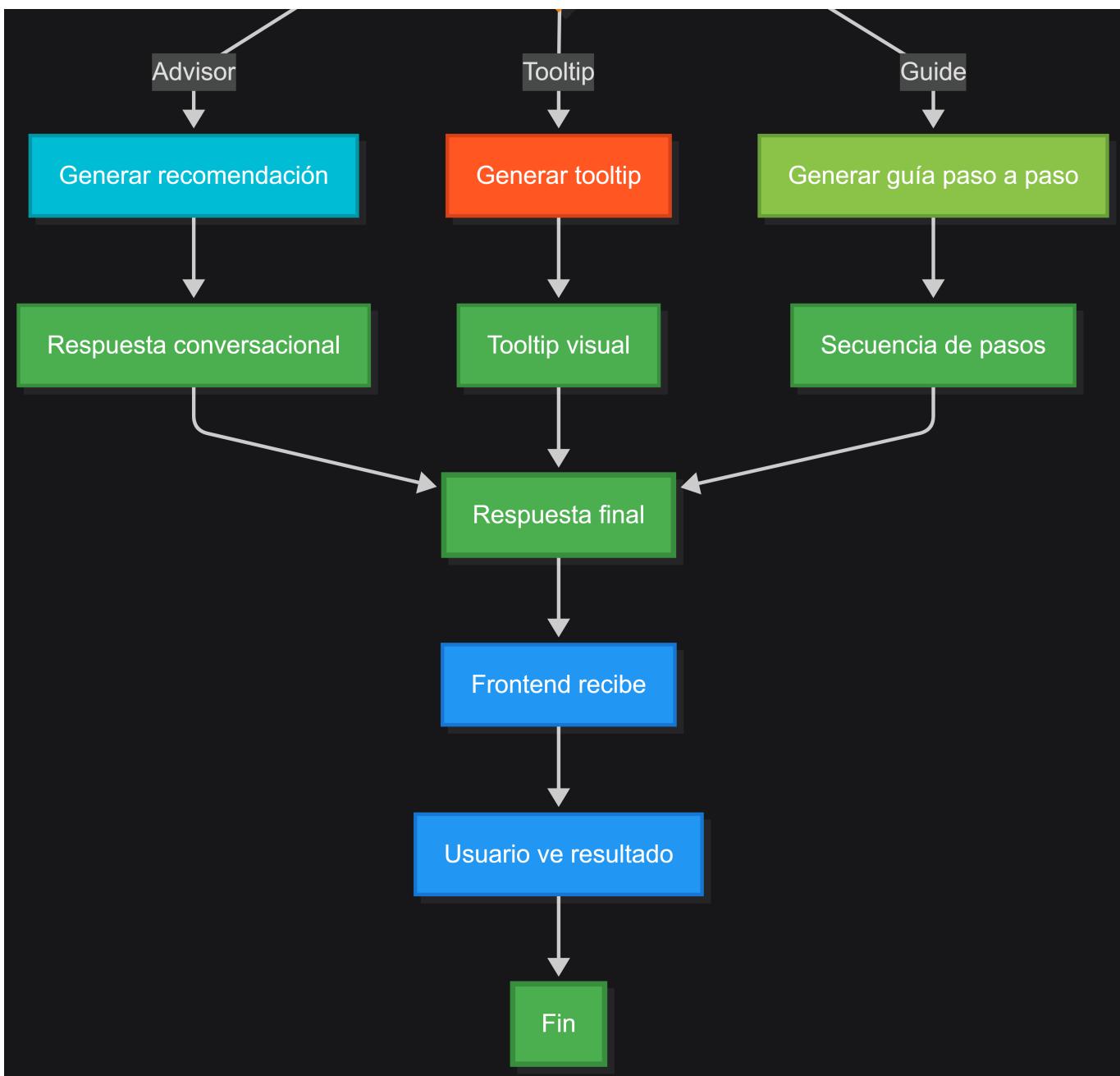
Note right of A: 💫 Validación de datos

A->>G: 💾 Procesa entrada

Note right of G: ☑ Análisis de contexto  
G->>R: ⚡ Clasifica intención  
Note right of R: 🤖 IA Clasificadora  
R->>N: 🏠 Dirige a nodo específico  
Note right of N: 💬 Procesamiento especializado  
N->>L: 🌟 Sigue el procesamiento  
Note right of L: 🧠 Procesamiento AI  
L->>N: 📱 Retorna respuesta  
Note right of N: ⚡ Formateo de respuesta  
N->>G: 📁 Actualiza estado  
Note right of G: 🗄 Persistencia de datos  
G->>A: 📊 Respuesta estructurada  
Note right of A: 🔑 Serialización JSON  
A->>F: 📄 JSON con respuesta  
Note right of F: 🎨 Renderizado UI  
F->>U: 📺 Muestra resultado  
Note right of U: ✅ Experiencia completa

## Diagrama de Flujo





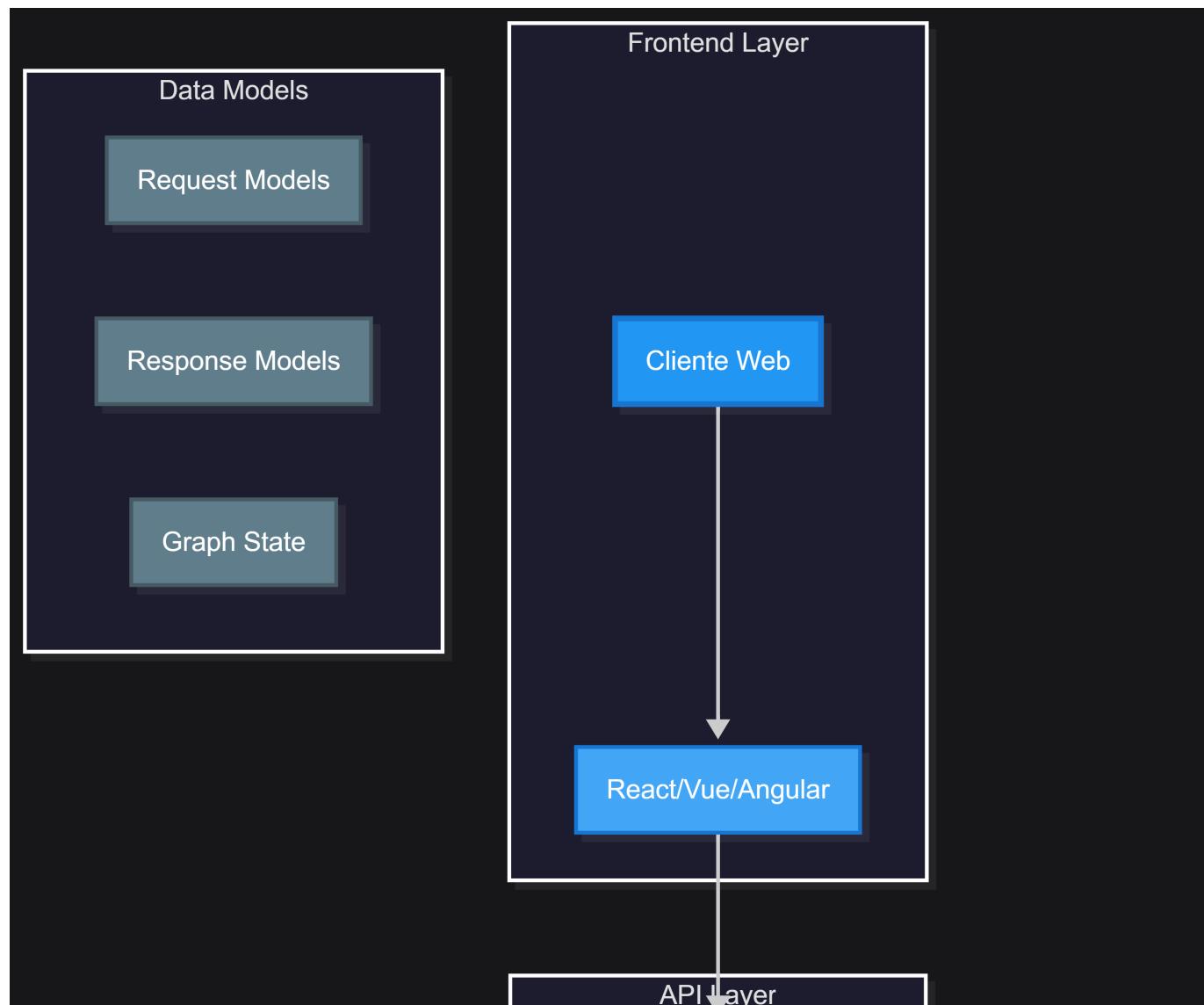
```

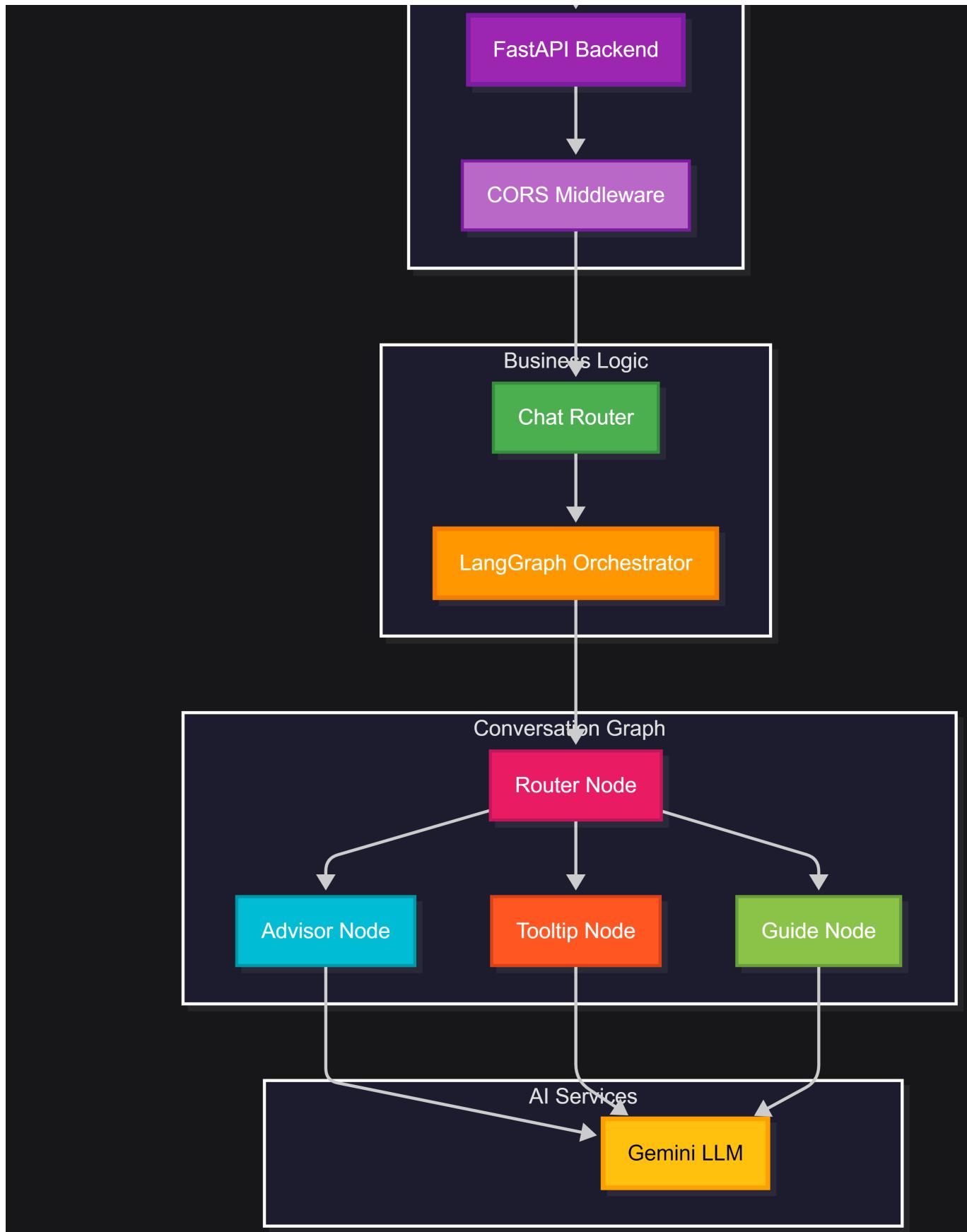
graph TD
    A[Inicio] --> B[Usuario envía solicitud]
    B --> C[Procesar mensaje]
    C --> D[Router clasifica intención]
    D --> E{{Tipo de asistencia?}}
    
    E -->|Advisor| F[Generar recomendación]
    E -->|Tooltip| G[Generar tooltip]
    E -->|Guide| H[Generar guía paso a paso]
    
    F --> I[Respuesta conversacional]
    G --> J[Tooltip visual]
    H --> K[Secuencia de pasos]
    
    I --> L[Respuesta final]
    J --> L
    K --> L
    L --> M[Frontend recibe]
    M --> N[Usuario ve resultado]
    N --> O[Fin]
  
```

L --> M[Frontend recibe]  
M --> N[Usuario ve resultado]  
N --> O[Fin]

```
style A fill:#4CAF50,stroke:#388E3C,stroke-width:3px,color:#ffffff  
style B fill:#2196F3,stroke:#1976D2,stroke-width:3px,color:#ffffff  
style C fill:#42A5F5,stroke:#1976D2,stroke-width:2px,color:#ffffff  
style D fill:#E91E63,stroke:#C2185B,stroke-width:3px,color:#ffffff  
style E fill:#FF9800,stroke:#F57C00,stroke-width:3px,color:#ffffff  
style F fill:#00BCD4,stroke:#0097A7,stroke-width:2px,color:#ffffff  
style G fill:#FF5722,stroke:#D84315,stroke-width:2px,color:#ffffff  
style H fill:#8BC34A,stroke:#689F38,stroke-width:2px,color:#ffffff  
style I fill:#4CAF50,stroke:#388E3C,stroke-width:2px,color:#ffffff  
style J fill:#4CAF50,stroke:#388E3C,stroke-width:2px,color:#ffffff  
style K fill:#4CAF50,stroke:#388E3C,stroke-width:2px,color:#ffffff  
style L fill:#4CAF50,stroke:#388E3C,stroke-width:3px,color:#ffffff  
style M fill:#2196F3,stroke:#1976D2,stroke-width:2px,color:#ffffff  
style N fill:#2196F3,stroke:#1976D2,stroke-width:2px,color:#ffffff  
style O fill:#4CAF50,stroke:#388E3C,stroke-width:3px,color:#ffffff
```

## Diagrama Arquitectura





```
graph TD
    subgraph "Frontend Layer"
        A[Cliente Web] --> B[React/Vue/Angular]
    end
```

```
graph TD
    subgraph "API Layer"
        B[FastAPI Backend] --> C[CORS Middleware]
        C --> D[LangGraph Orchestrator]
    end

    subgraph "Business Logic"
        D --> E[Chat Router]
        E --> G[LangGraph Orchestrator]
    end

    subgraph "Conversation Graph"
        G --> I[Router Node]
        I --> J[Advisor Node]
        I --> K[Tooltip Node]
        I --> L[Guide Node]
    end

    subgraph "AI Services"
        J --> M[Gemini LLM]
        K --> M
        L --> M
    end

    subgraph "Data Models"
        O[Request Models]
        P[Response Models]
        Q[Graph State]
    end

    style A fill:#2196F3,stroke:#1976D2,stroke-width:3px,color:#ffffff
    style B fill:#42A5F5,stroke:#1976D2,stroke-width:2px,color:#ffffff
    style C fill:#9C27B0,stroke:#7B1FA2,stroke-width:3px,color:#ffffff
    style D fill:#BA68C8,stroke:#7B1FA2,stroke-width:2px,color:#ffffff
    style E fill:#4CAF50,stroke:#388E3C,stroke-width:2px,color:#ffffff
    style G fill:#FF9800,stroke:#F57C00,stroke-width:3px,color:#ffffff
    style I fill:#E91E63,stroke:#C2185B,stroke-width:2px,color:#ffffff
    style J fill:#00BCD4,stroke:#0097A7,stroke-width:2px,color:#ffffff
    style K fill:#FF5722,stroke:#D84315,stroke-width:2px,color:#ffffff
    style L fill:#8BC34A,stroke:#689F38,stroke-width:2px,color:#ffffff
    style M fill:#FFC107,stroke:#FFA000,stroke-width:3px,color:#000000
    style O fill:#607D8B,stroke:#455A64,stroke-width:2px,color:#ffffff
    style P fill:#607D8B,stroke:#455A64,stroke-width:2px,color:#ffffff
    style Q fill:#607D8B,stroke:#455A64,stroke-width:2px,color:#ffffff
```

## API Endpoints

### Chat Endpoint

**POST /chat**

Procesa mensajes del usuario y retorna asistencia contextual.

**Cuerpo de la solicitud:**

```
{  
    "userInput": "¿Cuál me recomiendas?",  
    "uiContext": {  
        "view": "product_list",  
        "visibleProducts": [...],  
        "cartItems": [...],  
        "searchTerm": "laptop"  
    }  
}
```

**Respuesta exitosa (200):**

```
{  
    "response": "Basándome en los productos que veo, te recomiendo la Laptop Pro X1  
por su excelente relación calidad-precio y las buenas reseñas que tiene."  
}
```

**Respuesta con tooltip (200):**

```
{  
    "response": "Te muestro dónde está el botón de agregar al carrito",  
    "popup": {  
        "type": "info",  
        "target": "product_button",  
        "title": "Aregar al Carrito",  
        "message": "Haz clic aquí para agregar el producto",  
        "targetInfo": {  
            "ID": 1  
        }  
    }  
}
```

## Estructura de Carpetas y Archivos

```
ecommerce-copilot-backend/  
    └── app/  
        ├── config/  
        │   └── config.py          # Configuración del sistema  
        ├── graph/  
        │   ├── graph_state.py    # Estado del grafo de conversación  
        │   ├── llm.py             # Configuración del LLM  
        │   ├── main_graph.py     # Grafo principal de conversación  
        │   └── nodes/  
        │       └── advisor_node.py # Nodo de asesoramiento
```

```

        └── guide_node.py      # Nodo de guías paso a paso
        └── router_node.py   # Nodo clasificador
        └── tooltip_node.py  # Nodo de tooltips

    └── models/
        ├── guide_output.py  # Modelo de salida para guías
        ├── request_model.py # Modelo de solicitud
        ├── router_output.py # Modelo de salida del router
        ├── tooltip_output.py# Modelo de salida para tooltips
        └── ui_context.py    # Modelo de contexto UI

    └── routes/
        ├── chat.py          # Endpoint de chat
        └── vision.py        # Endpoint de visión

    └── services/
        └── gemini_client.py # Cliente de Gemini AI

docs/
└── README.pdf           # Documentación en PDF

image/
└── diagrama-arquitectura-ecommerce-copilot-backend.png
└── diagrama-caso.uso-copilot-backend.png
└── diagrama-flujo-ecommerce-copilot-backend.png
└── diagrama-secuencia-copilot-backend.png
└── flow.png              # Diagrama del grafo generado

logger.py                # Configuración de logging
main.py                  # Aplicación principal FastAPI
README.md                # Documentación del proyecto
render.yaml              # Configuración de despliegue
requirements.txt          # Dependencias del proyecto

```

## FAQ's y Problemas Comunes

¿Cómo configurar la API key de Gemini?

1. Ve a [Google AI Studio](#)
2. Crea una nueva API key
3. Agrega la key al archivo `.env`:

GEMINI\_API\_KEY=tu\_api\_key\_aqui

¿Cómo personalizar las respuestas del asistente?

- Modifica los mensajes del sistema en los archivos de nodos (`adviser_node.py`, `tooltip_node.py`, etc.)
- Ajusta la lógica de clasificación en `router_node.py`
- Personaliza los modelos de salida en la carpeta `models/`

¿Cómo agregar nuevos tipos de asistencia?

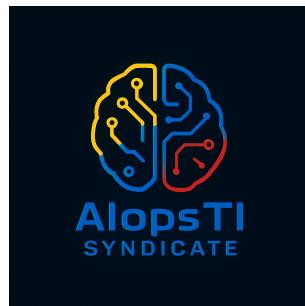
1. Crea un nuevo nodo en `app/graph/nodes/`
2. Agrega el nodo al grafo en `main_graph.py`

3. Actualiza la lógica de clasificación en `router_node.py`
4. Crea el modelo de salida correspondiente

## ¿Cómo desplegar en producción?

El proyecto está configurado para despliegue automático en Render.com:

1. Conecta tu repositorio a Render
2. Configura las variables de entorno en Render
3. El despliegue se realizará automáticamente



**Desarrollado por el** Equipo de Desarrollo AIopsTI Syndicate