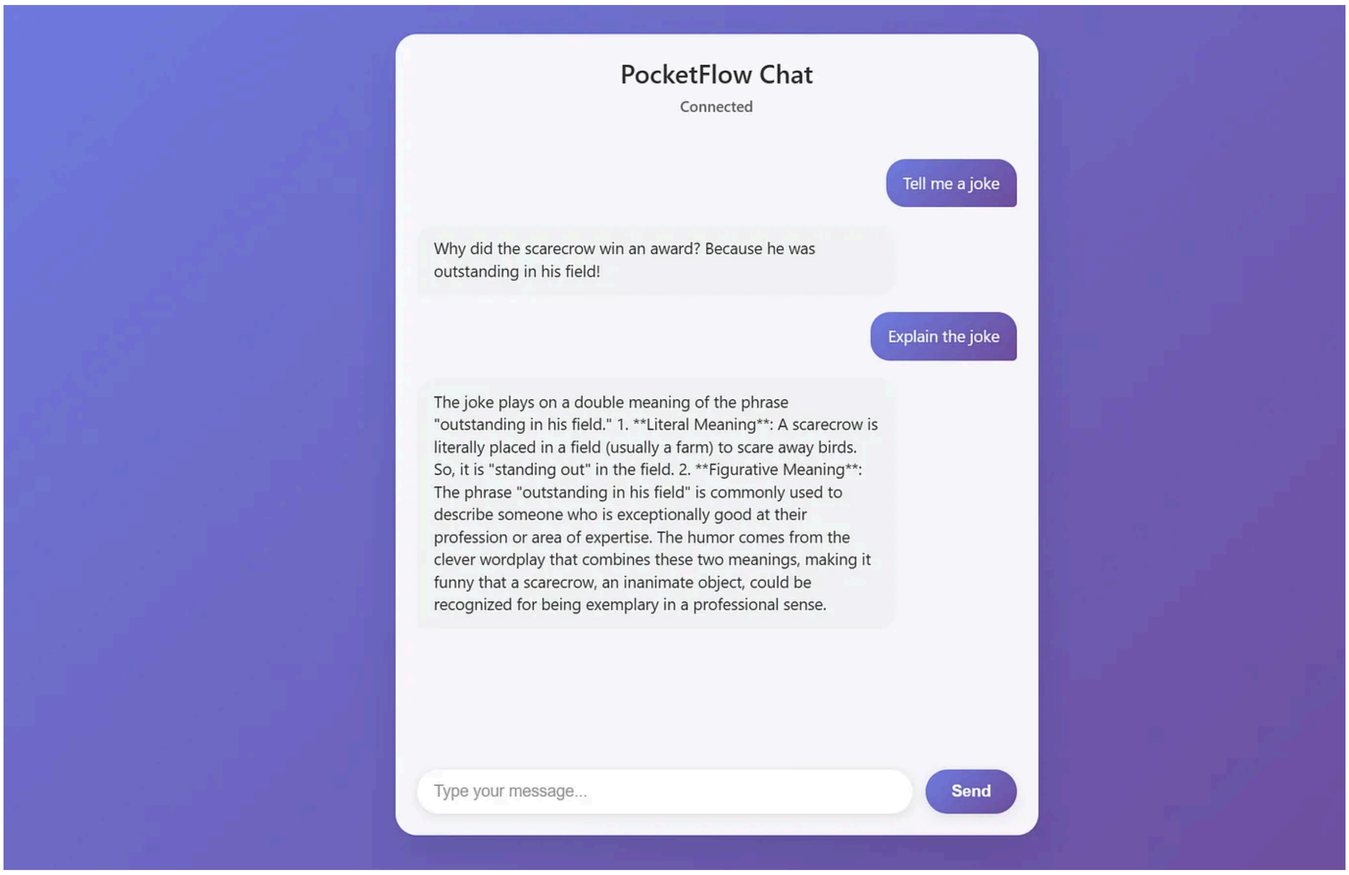
Build an LLM Web App in Python from Scratch: Part 3 (FastAPI & WebSockets)

ZACHARY HUANG
JUN 08, 2025

5 1 S



Tell me a joke

Why did the scarecrow win an award? Because he was outstanding in his field!

Explain the joke

The joke plays on a double meaning of the phrase "outstanding in his field." 1. **Literal Meaning**: A scarecrow is literally placed in a field (usually a farm) to scare away birds. So, it is "standing out" in the field. 2. **Figurative Meaning**: The phrase "outstanding in his field" is commonly used to describe someone who is exceptionally good at their profession or area of expertise. The humor comes from the clever wordplay that combines these two meanings, making it funny that a scarecrow, an inanimate object, could be recognized for being exemplary in a professional sense.

> *Ever watched ChatGPT type back to you word by word, like it's actually thinking out l*
> *That's* ***streaming AI*** *in action, and it makes web apps feel incredibly alive! Today, we'r*
> *building exactly that: a real-time* ***AI chatbot web app*** *where responses flow in instantly*
> *more staring at loa*
> ***WebSockets*** *for live* *m*
> *your web app feel li* *par*
> *the [FastAPI WebSc]()*

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

1. Why Your AI Web App Should Stream (It's a Game Changer!) 🚀

Picture this: You ask an AI a question, then... you wait. And wait. Finally, BOOM wall of text appears all at once. Feels clunky, right?

Now imagine this instead: You ask your question, and the AI starts "typing" back immediately – word by word, just like texting with a friend. **That's the magic of streaming for AI web apps.**

Why streaming rocks: It feels lightning fast, keeps users engaged, and creates nat conversation flow. No more "is this thing broken?" moments!

We're creating a **live AI chatbot web app** that streams responses in real-time. You type a message, and watch the AI respond word by word, just like the pros do it.

Our toolkit:

- 🔧 **FastAPI** – Blazing fast Python web framework
- 🔧 **WebSockets** – The secret sauce for live, two-way chat
- 🔧 PocketFlow – Our LLM framework in 100 lines

Quick catch-up on our series:

- Part 1: Built command-line AI tools ✅
- Part 2: Created interactive web apps with Streamlit ✅
- **Part 3 (You are l**
- **Part 4 (Coming**

Want to see streamir ur simpler guide: "Streaming LLM Responses — Tutorial For Dummies".

Ready to make your AI web app feel like magic? Let's dive in!

2. FastAPI + WebSockets = Real-Time Magic ⚡

To build our streaming chatbot, we need two key pieces: **FastAPI** for a blazing-fas
backend and **WebSockets** for live, two-way chat.

FastAPI: Your Speed Demon Backend

FastAPI is like the sports car of Python web frameworks – fast, modern, and asyn
ready. Perfect for AI apps that need to handle multiple conversations at once.

Most web apps work like old-school mail: Browser sends request → Server proces
Sends back response → Done. Here's a basic FastAPI example:

```
from fastapi import FastAPI
app = FastAPI()

@app.get("/hello")
async def say_hello():
    return {"greeting": "Hi there!"}
```

What's happening here?

- `app = FastAPI()` – Creates your web server
- `@app.get("/hello")` – Says "when someone visits `/hello`, run the functi
below"
- `async def sa`
- `return {"gre`
browser

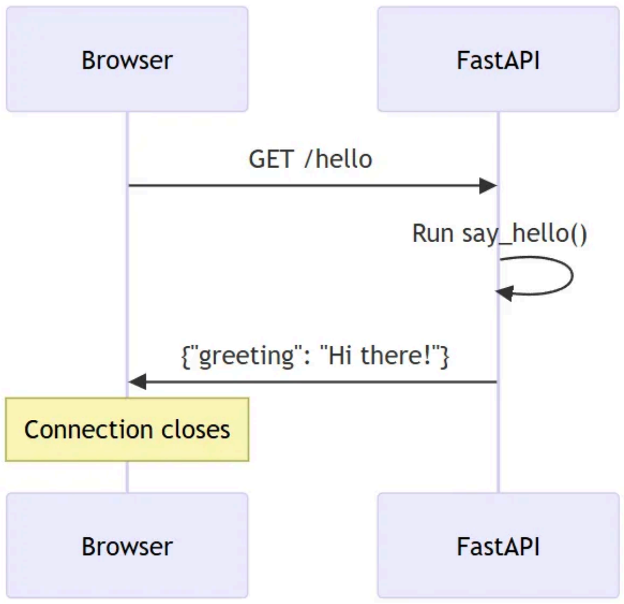
Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

When you visit `http://localhost:8000/hello`, you'll see `{"greeting": '` `there!"}` in your browser!

Your First FastAPI App Flow:



Simple enough, but for chatbots we need something more interactive...

WebSockets: Live Chat Superpowers

WebSockets turn your web app into a live phone conversation. Instead of sending messages back and forth, you open a connection that stays live for instant back-ar forth chat.

Here's a simple echo server that repeats whatever you say:

```
from fastapi import FastAPI, WebSocket

app = FastAPI()

@app.websocket("
async def chat_e
    await websoc
    while True:
```

Looks like an article worth saving!

Option Q

Hover over the brain icon or use hotkeys to save with Memex.

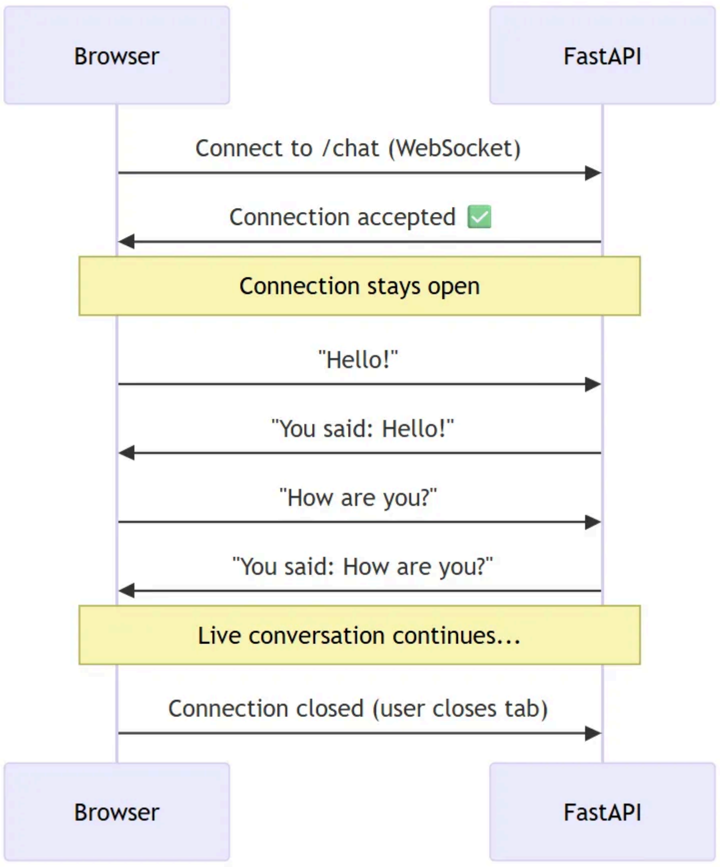
Remind me later Hide Forever

```
        message = await websocket.receive_text() # Listen
        await websocket.send_text(f"You said: {message}") # Reply
```

The browser side is just as simple:

```
<input id="messageInput" placeholder="Say something..."/>
<button onclick="sendMessage()">Send</button>
<div id="chatLog"></div>

<script>
    const ws = new WebSocket("ws://localhost:8000/chat");
    const chatLog = document.getElementById('chatLog');

    ws.onmessage = (event) => {
        chatLog.innerHTML += `<p>Server: ${event.data}</p>`;
    };

    function sendMessage() {
        const message = document.getElementById('messageInput').value
        ws.send(message);
        chatLog.innerHTML += `<p>You: ${message}</p>`;
        document.getElementById('messageInput').value = '';
    }
</script>
```

WebSocket Chat Flow:



That's it! You now have live, real-time communication between browser and serve
Perfect foundation for our streaming AI chatbot!

3. Adding AI to the Mix: Why Async Matters 🤖

Great! We have live chat working. But here's the thing: calling an AI like ChatGP
takes time (sometimes 3-5 seconds). If our server just sits there waiting, our whole
app freezes. Not good!

The problem: Normal code is like a single-lane road. When the AI is thinking,
everything else stops.

The solution: Async code is like a highway with multiple lanes. While AI is thinki
in one lane, other us

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

PocketFlow G

Remember PocketFl ta
into simple steps. For web apps, we need the async version:

- `AsyncNode` – Each step can wait for AI without blocking others
- `AsyncFlow` – Manages the whole conversation workflow

Here's the magic difference:

```
# ❌ This blocks everything
def call_ai(message):
    response = openai.chat.completions.create(...)  # Everyone waits!
    return response

# ✅ This lets others keep chatting
async def call_ai_async(message):
    response = await openai.chat.completions.create(...)  # Just this
task waits
    return response
```

Streaming Chat Node: The Star of the Show

Our `StreamingChatNode` does three things:

1. **Prep:** Add user message to chat history
2. **Execute:** Call AI and stream response word-by-word via WebSocket
3. **Post:** Save AI's complete response to history

```
class StreamingChatNode(AsyncNode):
    async def prep_async(self, shared):
        # Add user message to history
        history = shared.get("conversation_history", [])
        history.append({"role": "user", "content":
shared["user_message"]})
        return h

    async def ex
        messages

        # Stream
        full_response = ""
```

Looks like an article worth saving!

Hover over the brain icon or use hotkeys to save with Memex.

Option Q

Remind me later Hide Forever

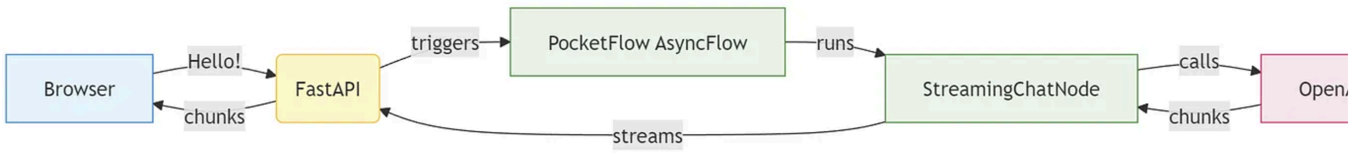
```
        async for chunk in stream_llm(messages):
            full_response += chunk
            await websocket.send_text(json.dumps({"content": chunk}))

        return full_response

    async def post_async(self, shared, prep_res, exec_res):
        # Save complete AI response
        shared["conversation_history"].append({
            "role": "assistant",
            "content": exec_res
        })
```

That's it! The node streams AI responses live while keeping chat history. Next, let see how this all connects together!

4. Putting It All Together: The Complete Streaming Flow 🔄

Time to connect all the pieces! Here's how a user message flows through our streaming chatbot:

The Journey of a Message:



User sends message → FastAPI receives it → PocketFlow handles AI logic → Strea response back live!

The FastAPI V

Looks like an article worth saving!

Option Q

Here's the main Fast

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

```
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    chat_memory = {
        "websocket": websocket,
        "conversation_history": []
    }

    try:
        while True:
            # Get user message
            user_data = await websocket.receive_text()
            message = json.loads(user_data)  # {"content": "Hello!"}
            chat_memory["user_message"] = message["content"]

            # Run our PocketFlow
            chat_flow = create_streaming_chat_flow()
            await chat_flow.run_async(chat_memory)

    except WebSocketDisconnect:
        print("User left the chat")

def create_streaming_chat_flow():
    return AsyncFlow(start_node=StreamingChatNode())
```

What happens:

1. Accept WebSocket connection
2. Wait for user messages in a loop
3. For each message, run our `StreamingChatNode`
4. The node handles AI calling + streaming automatically!

Note: Each WebSocket connection gets its own `chat_memory` dictionary with th connection, latest message, and full conversation history. This lets each user have independent convers

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

Frontend: The

On the browser side,

```
<div id="aiResponse"></div>
<input id="userInput" placeholder="Type your message..."/>
<button onclick="sendMessage()">Send</button>

<script>
const ws = new WebSocket("ws://localhost:8000/ws");
const aiResponse = document.getElementById("aiResponse");

// The magic: append each chunk as it arrives
ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    if (data.content) {
        aiResponse.textContent += data.content; // Stream word by wor
    }
};

function sendMessage() {
    const input = document.getElementById("userInput");
    aiResponse.textContent = ""; // Clear for new response
    ws.send(JSON.stringify({content: input.value}));
    input.value = "";
}
</script>
```

The streaming happens in `ws.onmessage` – each time the server sends a text ch
we append it to the display. That's how you get the "typing" effect!

Pretty neat, right? You now have all the pieces for a real-time streaming AI chatb

5. Mission Accomplished! You Built a Real-Tim AI Chatbot 🎉

Boom! You just built m
waiting around – you

Looks like an article worth saving! Option Q

Hover over the brain icon or use hotkeys to save with Memex.

Remind me later Hide Forever

What you crushed t

- ⚡ **FastAPI + W**

- 🔄 **Async PocketFlow** – AI calls that don't freeze your app
- 🚀 **Streaming responses** – Watch the AI "type" in real-time

You've officially joined the ranks of developers building modern, responsive AI w apps. Pretty cool, right?

What's next in our series:

- Part 1: Command-line AI tools ✅
- Part 2: Interactive web apps with Streamlit ✅
- **Part 3 (You just finished!)**: Real-time streaming ✅
- **Part 4 (Coming up!)**: Background tasks for heavy AI work

Ready for the big leagues? Part 4 will tackle those marathon AI tasks – think generating reports or complex analyses that take minutes, not seconds. We'll expl background processing and Server-Sent Events to keep users happy even during tl heavy lifting.

Want to try this yourself? *Grab the complete code from the PocketFlow cookbook: Fast. WebSocket Chat Example You're building some serious AI web development skills! See y Part 4!* 🚀

Thanks for reading Pocket Flow! Subscribe for free to receive new posts and support my work.

5 Lik

Looks like an article worth saving!

Option Q

Hover over the brain icon or use hotkeys to save with Memex.

← Previous

Remind me later

Hide Forever

ext

Discussion about this post

Comments Restacks

Write a comment...