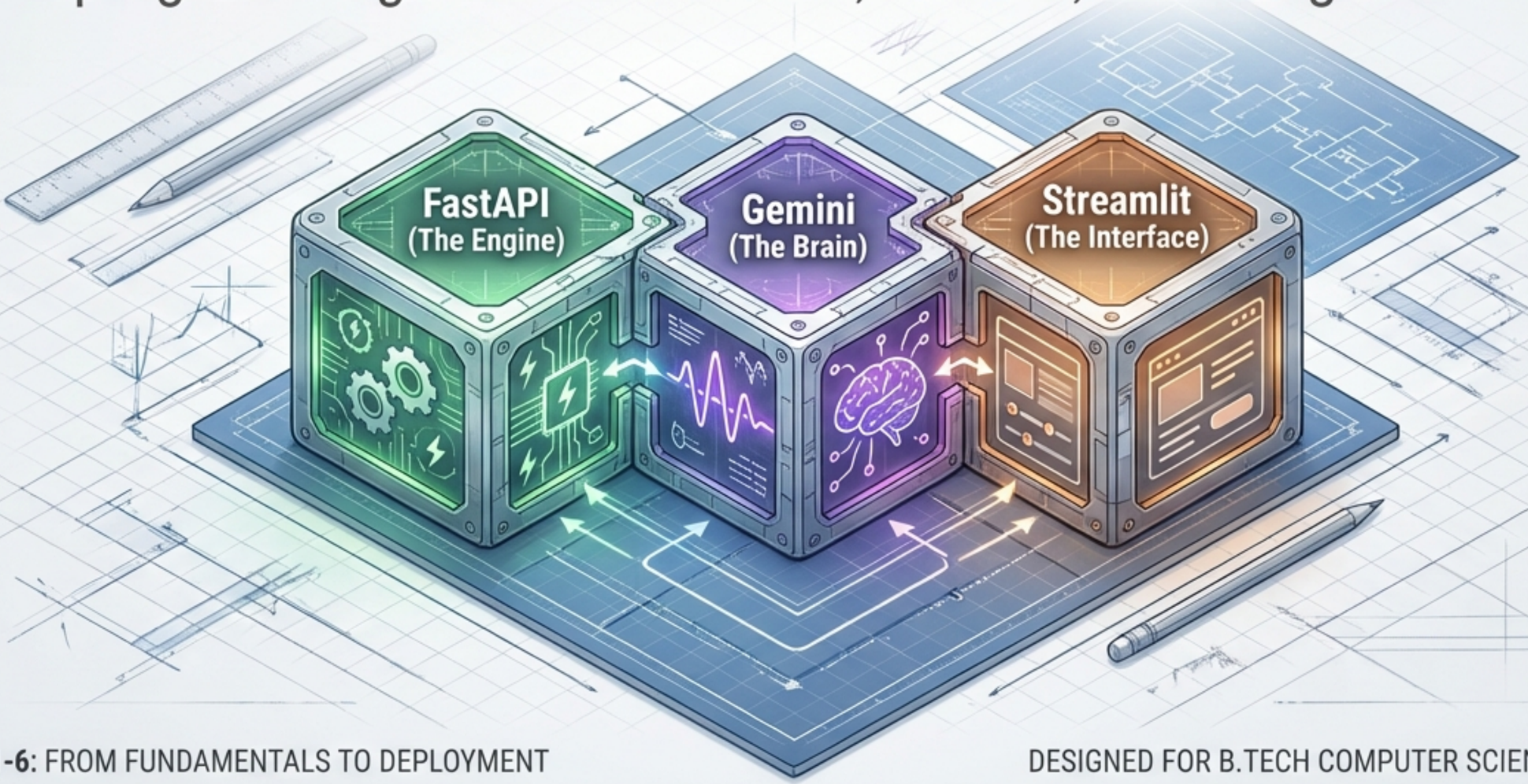
Building GenAI Applications: From Zero to API

A progressive guide to REST APIs, FastAPI, and Google Gemini



MODULE 1-6: FROM FUNDAMENTALS TO DEPLOYMENT

DESIGNED FOR B.TECH COMPUTER SCIENCE STUDENTS

The Journey of a Request

Key Concept: This entire journey happens through REST APIs (Representational State Transfer).

The Language of the Web: HTTP & JSON

The Four Essential Verbs

GET
Retrieve Data
(e.g., Get chat history)

POST
Send/Create Data
(e.g., Send prompt to AI)

PUT
Update Data
(e.g., Change settings)

DELETE
Remove Data
(e.g., Clear history)

The Scorecard

- 🟢 **200 OK**: Success
- 🟠 **400 Bad Request**: Invalid Input
- 🔴 **401 Unauthorized**: Missing Key
- 🔴 **500 Internal Error**: Server Crash

Why JSON?

JSON is the **industry standard**. Lightweight, human-readable, and language-agnostic.

```
{
  "prompt": "Hello AI",
  "model": "gemini-flash"
}
```

The Engine: Why FastAPI?

Speed: High performance, on par with NodeJS and Go.

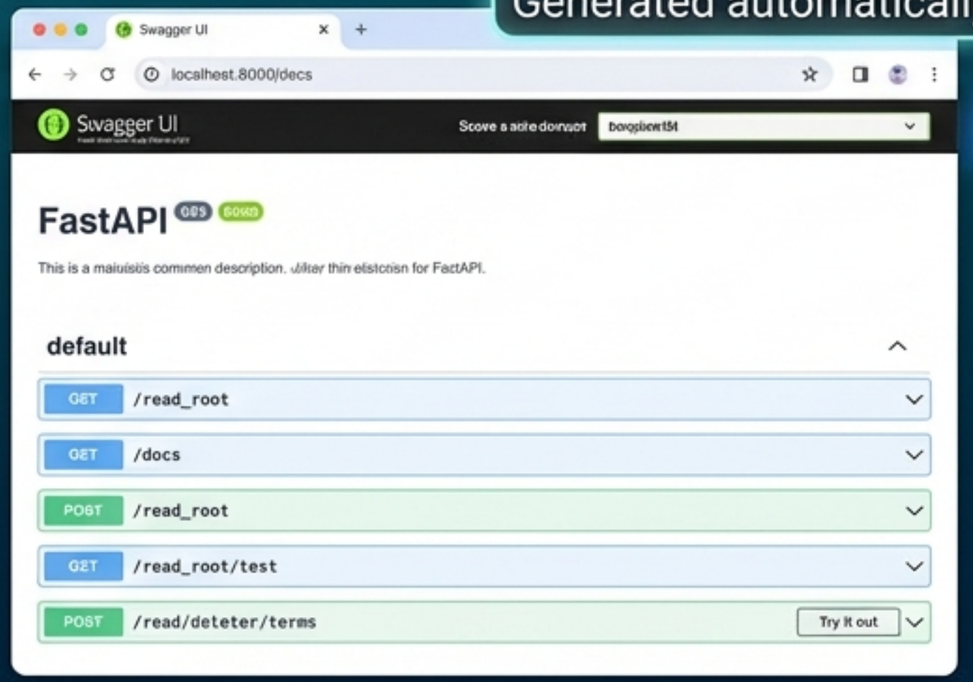
Async Native: Built for concurrency – crucial for waiting on AI responses.

Type Safe: Catches errors before the code even runs.

The Magic: Automatic interactive documentation.

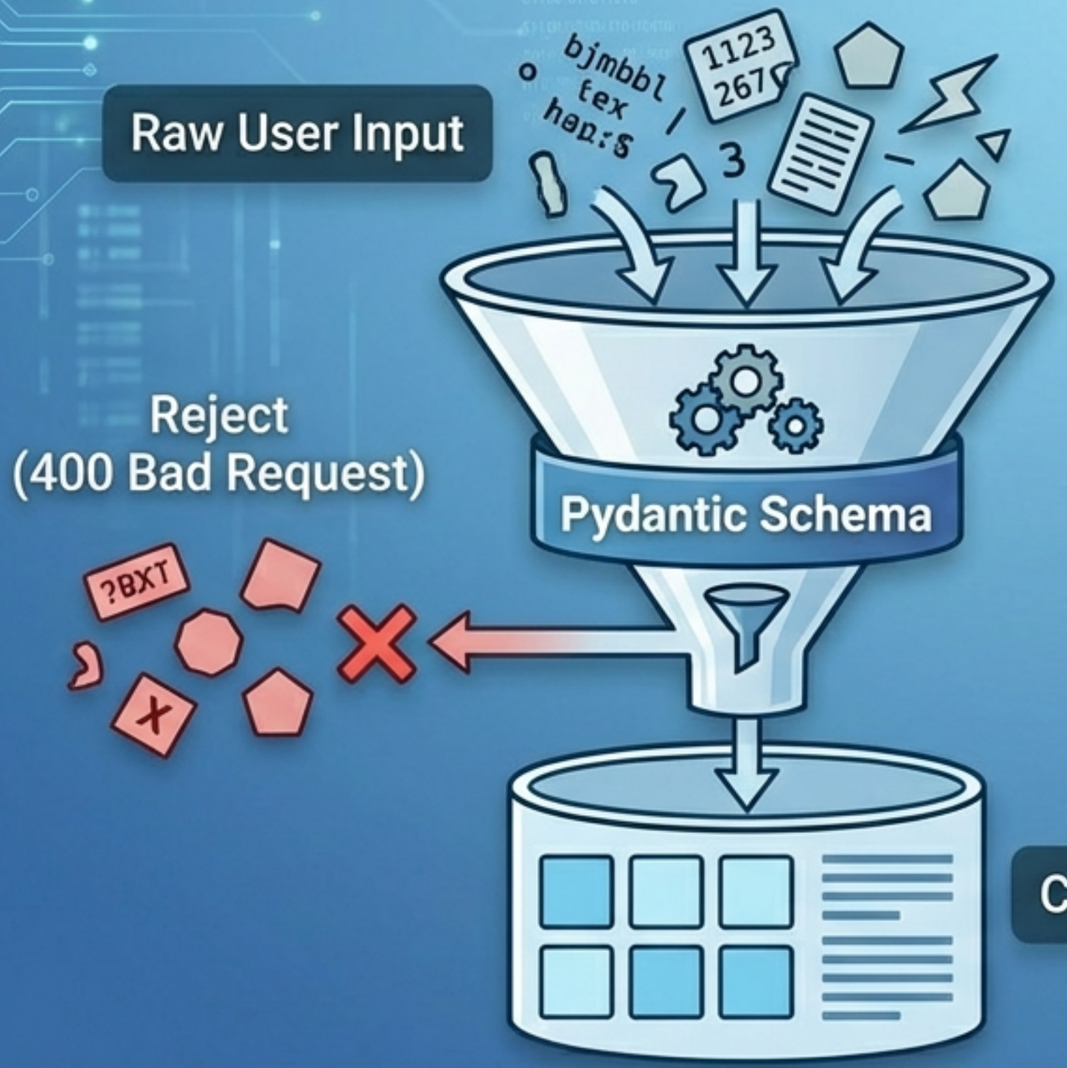
```python
app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}
```

Generated automatically!



Guardrails & Validation with Pydantic

Ensuring data integrity before it reaches the AI.




```python
from pydantic import BaseModel, Field

class ChatRequest(BaseModel):
    # We ensure the prompt is at least 1 character
    prompt: str = Field(..., min_length=1)
    model: str = "gemini-3-flash-preview"
```

Why Pydantic?

1. Auto-validation.
2. Editor Autocomplete.
3. Clear API Schemas.

Preparing the Brain: Setup & Security

Prerequisites

1. 🔑 Get **API Key** from Google AI Studio.

2. >_ Install library:
   `pip install langchain-google-genai`

THE GOLDEN RULE: NEVER hardcode API keys in your code!

The Wrong Way:

```
api_key = "AIzaSyD..." # ❌ DANGER
```

The Right Way:

```
api_key = os.getenv("GOOGLE_API_KEY")
# ✅ SAFE
```


```
import os
from dotenv import load_dotenv

load_dotenv()
api_key = os.getenv("GOOGLE_API_KEY")
```

Secure Environment Variable Loading

The First Neuron: Connecting Gemini

```python
from langchain_google_genai import ChatGoogleGenerativeAI

# 1. Initialize the Model
model = ChatGoogleGenerativeAI(model='gemini-3-flash-preview')

@app.post("/chat")
def chat(request: ChatRequest):
    # 2. The Trigger
    # The .invoke() method sends the prompt to Google
    response = model.invoke(request.prompt)

    # 3. The Response
    return {"response": response.content}
```

This single line contains the magic. It sends your text to Google's servers and awaits the intelligence.

Handling Errors Gracefully

Real-world apps don't just crash. They tell the user what went wrong.

Block 1 (The Check)

```
Is prompt empty?
```

↓ True

```
Raise HTTPEXCeption(400,
'Empty prompt')
```

Block 2 (The Try)

```
try:
    model.invoke()
```

Block 3 (The Catch)

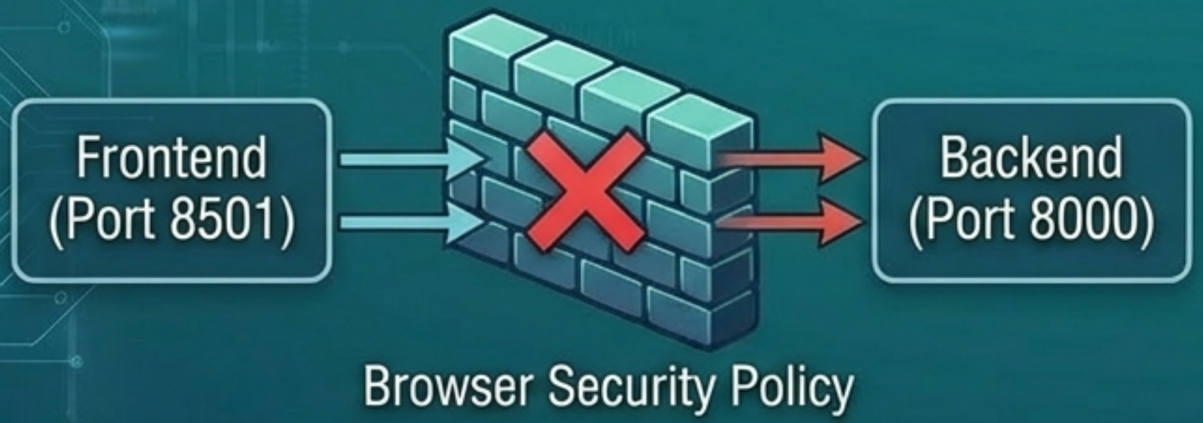
```
except Exception as e:
```

↓ Catch

```
Raise HTTPEXCeption(500,
'Server Error')
```

SAFETY NET
(EXCEPTION HANDLER)

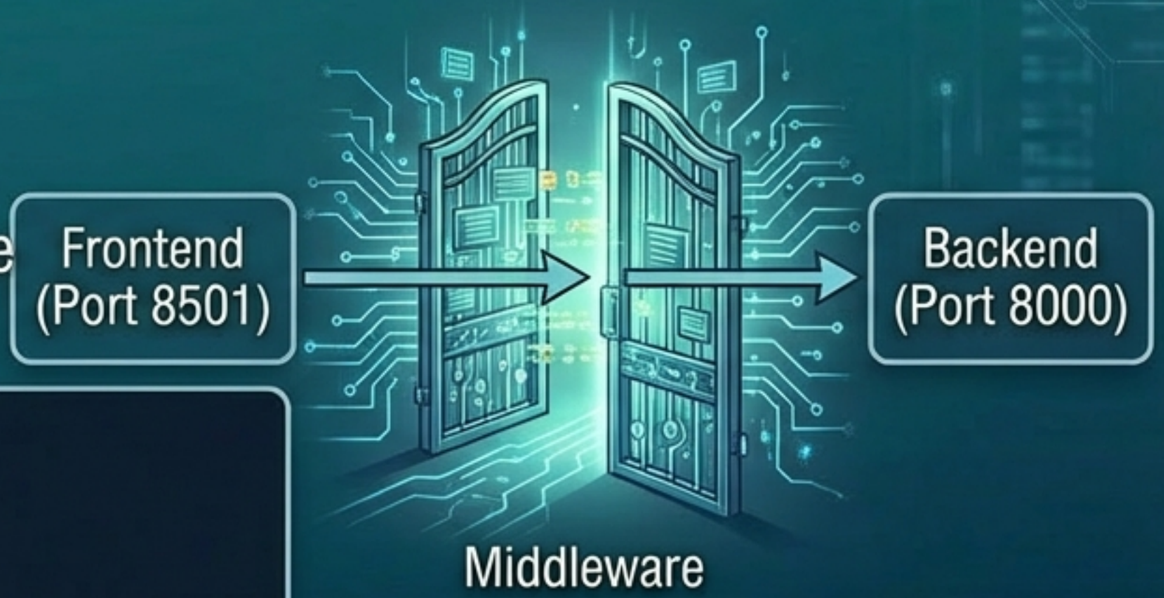
Bridging Ports: Understanding CORS



Browser Security Policy

The Solution:

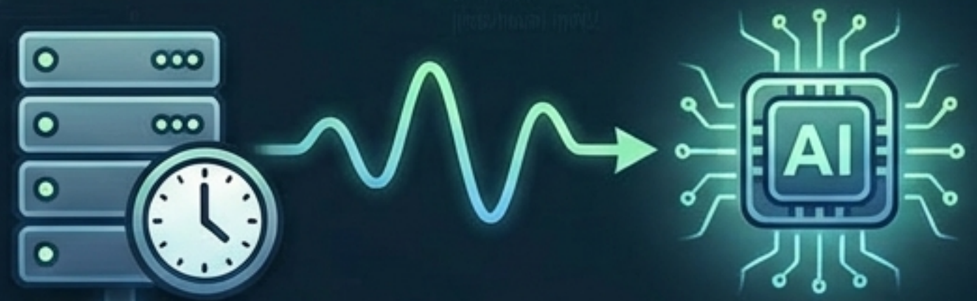
Cross-Origin Resource Sharing (CORS) Middleware allows the browser to trust the backend.




```python
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"], # Allows all origins (Development only)
    allow_methods=["*"], # Allows GET, POST, etc.
    allow_headers=["*"],
)
```

Memory & Asynchronous Support

Asynchronous Support



Async prevents the server from freezing while waiting for AI.

```
async def chat(...):
    await model.ainvoke(...)
```

Memory Concept

HumanMessage (User Input)

AIMessage (Bot Reply)

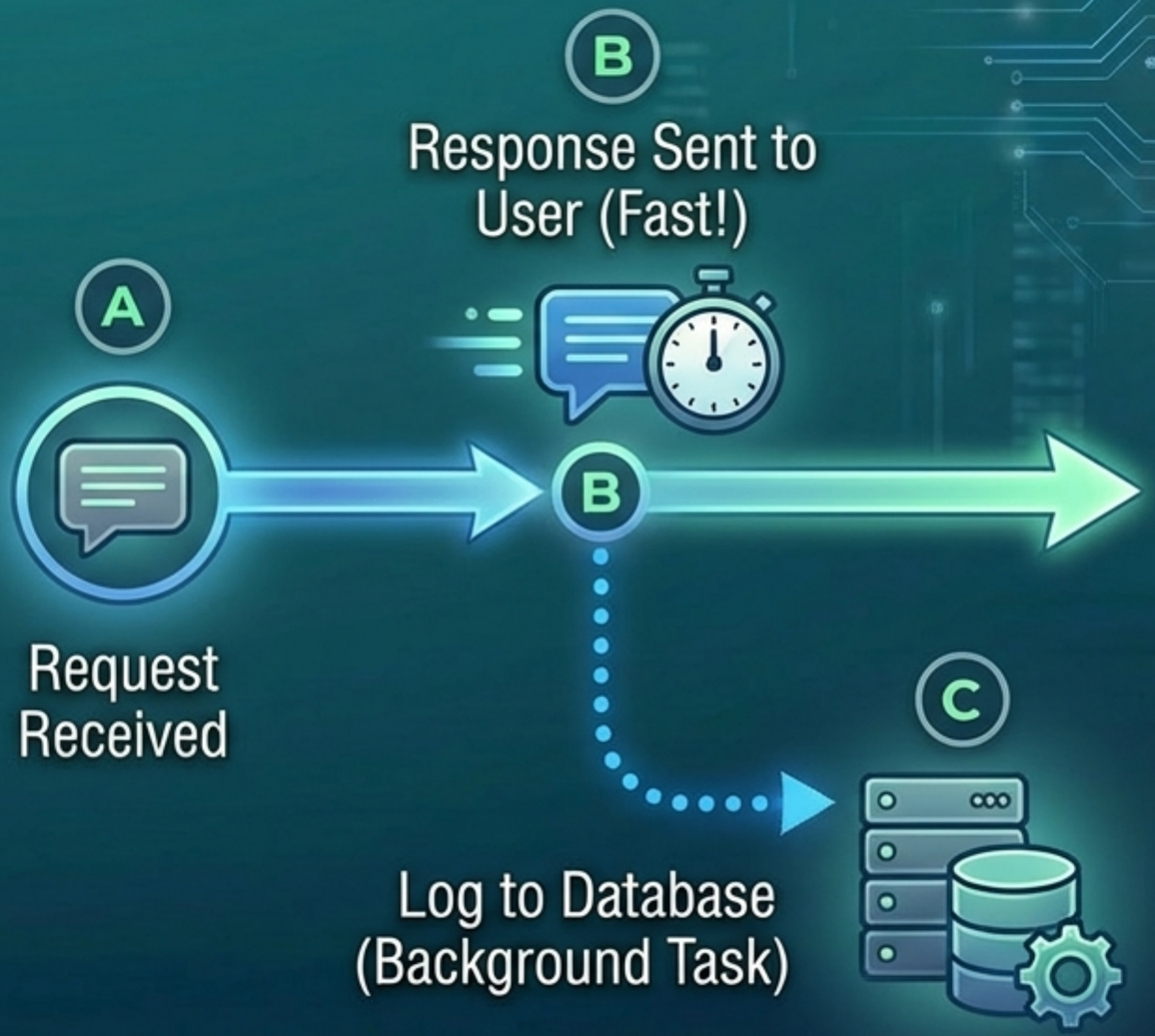
HumanMessage (Follow-up)

We store these messages in a list and send the *entire* history back to the model for context.

Behind the Scenes: Background Tasks

Concept: Keep the chat snappy.
Do the heavy lifting (logging, database saves) after the response is sent.

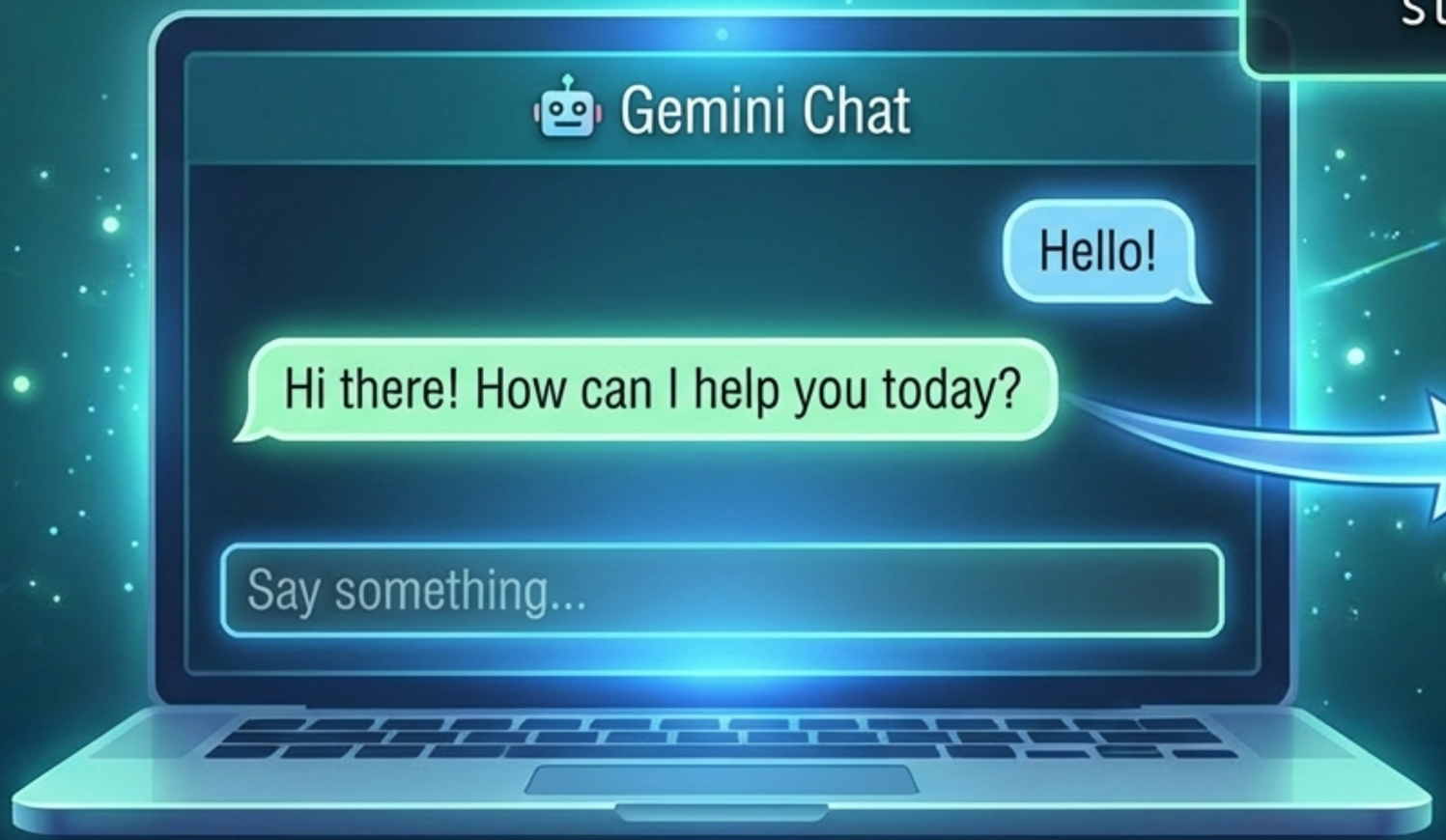
```python
from fastapi import BackgroundTasks

@app.post("/chat")
async def chat(req: ChatRequest,
               bg_tasks: BackgroundTasks):
    response = model.invoke(req.prompt)

    # This runs AFTER the return statement
    bg_tasks.add_task(log_conversation,
                      req.prompt, response)

    return {"response": response.content}
```



The Frontend Face: Streamlit

Build beautiful data apps in pure Python.
No HTML/CSS required.

```python
import streamlit as st

with st.chat_message("user"):
    st.write("Hello!")
```



`st.session_state` keeps this persistent.

The Handshake: Connecting UI to API

1. User types in `st.chat_input`.
2. Frontend sends JSON via `requests.post()`.
3. Backend processes and returns JSON.
4. Frontend displays result via `st.markdown()`.

```
import requests

API_URL = "http://localhost:8000/chat"

if prompt := st.chat_input("Ask anything"):
    # The Handshake
    response = requests.post(API_URL,
                             json={"prompt": prompt})

    if response.status_code == 200:
        st.write(response.json()"response"])
```

Professional Standards & Best Practices

my-genai-app/

main.py (The App)

models.py (Schemas)

.env (Secrets)

.gitignore (Config)

requirements.txt
(Dependencies)

Security:
Add `.env` and `__pycache__` to `.gitignore`.
Never push secrets to GitHub.

Reproducibility:
Always freeze dependencies:
`pip freeze > requirements.txt`

Your Roadmap to GenAI Mastery

The Stack Recap

FastAPI
The Engine
(Speed & Validation)

LangChain/Gemini
The Brain
(Intelligence)

Streamlit
The Face
(Interaction)

Learning Path Timeline

Week 1:
REST Fundamentals

Week 2:
Gemini Integration

Week 3:
Full Stack Deployment

Resources
Docs: fastapi.tiangolo.com | docs.streamlit.io
API: ai.google.dev

"Start simple. Test often in /docs. Build something amazing."