PyCon Italia 2026

Building a RAG from the ground up

with Hexagonal Architecture

Organise your code to build scalable AI applications

Giunio De Luca

AvenueIT SRL

DATE May 29, 2026

VENUE Savoia Regency Hotel · Bologna

Agenda

01 **How to query an LLM**

From naive prompts to context-aware queries

02 **What is a RAG**

Features to automate context driven responses

03 **Hexagonal architecture**

Ports, adapters, and clean boundaries

04 **Coding the RAG**

From dummies to a production-ready service

05 **Live demo**

Watch it run

Who am I?

- Born and raised in **Basilicata, Italy**
- Living in **Brussels, Belgium**
- Working as contractor at **AvenueIT SRL**
- PhD in Energetics Engineering **Paris-Saclay University**
- Author of **FastAPI Cookbook (Packt)**

Who are you?

Quick show of hands

< 3

years of
professional coding

3–6

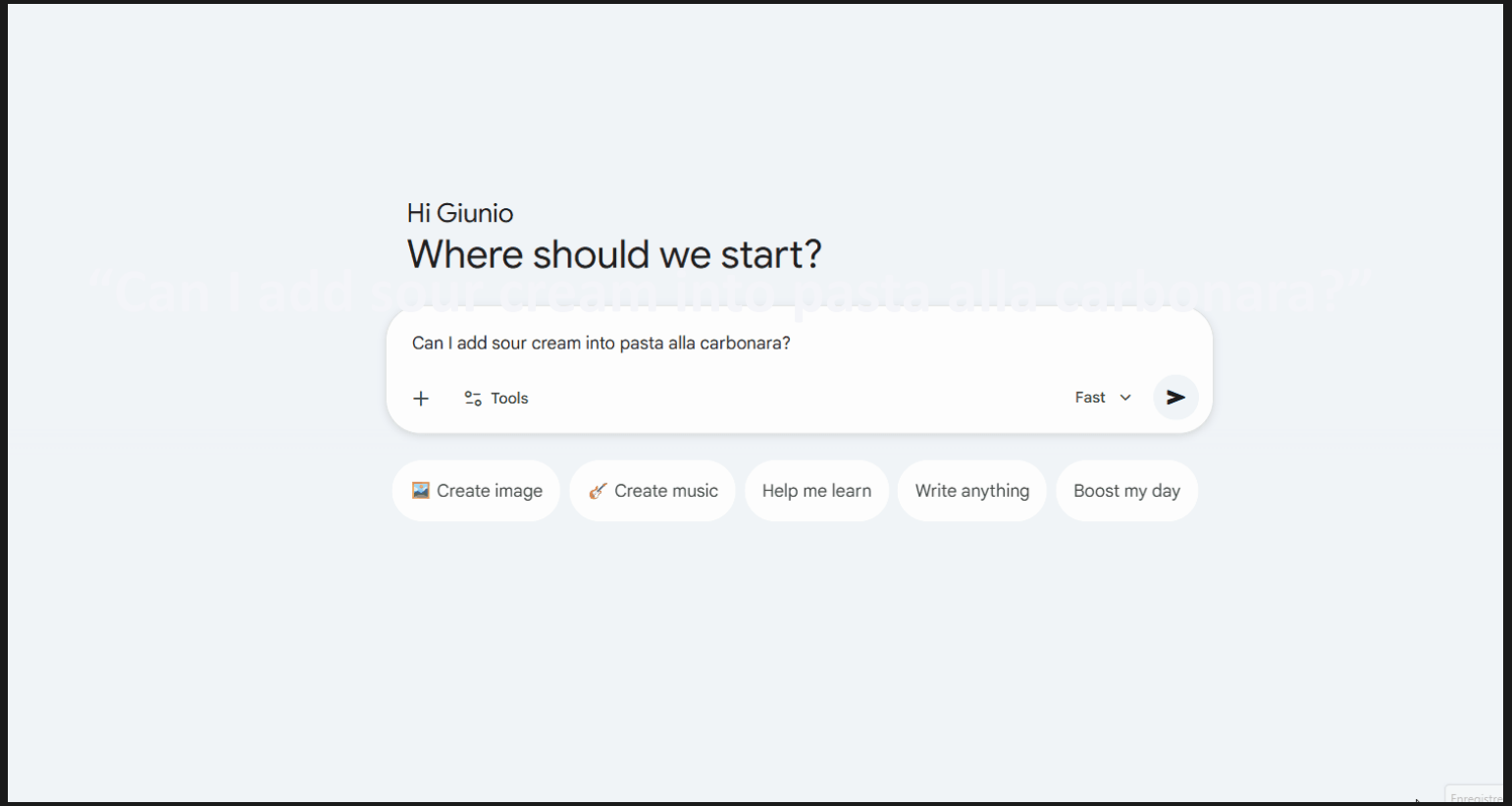
years
of experience

6+

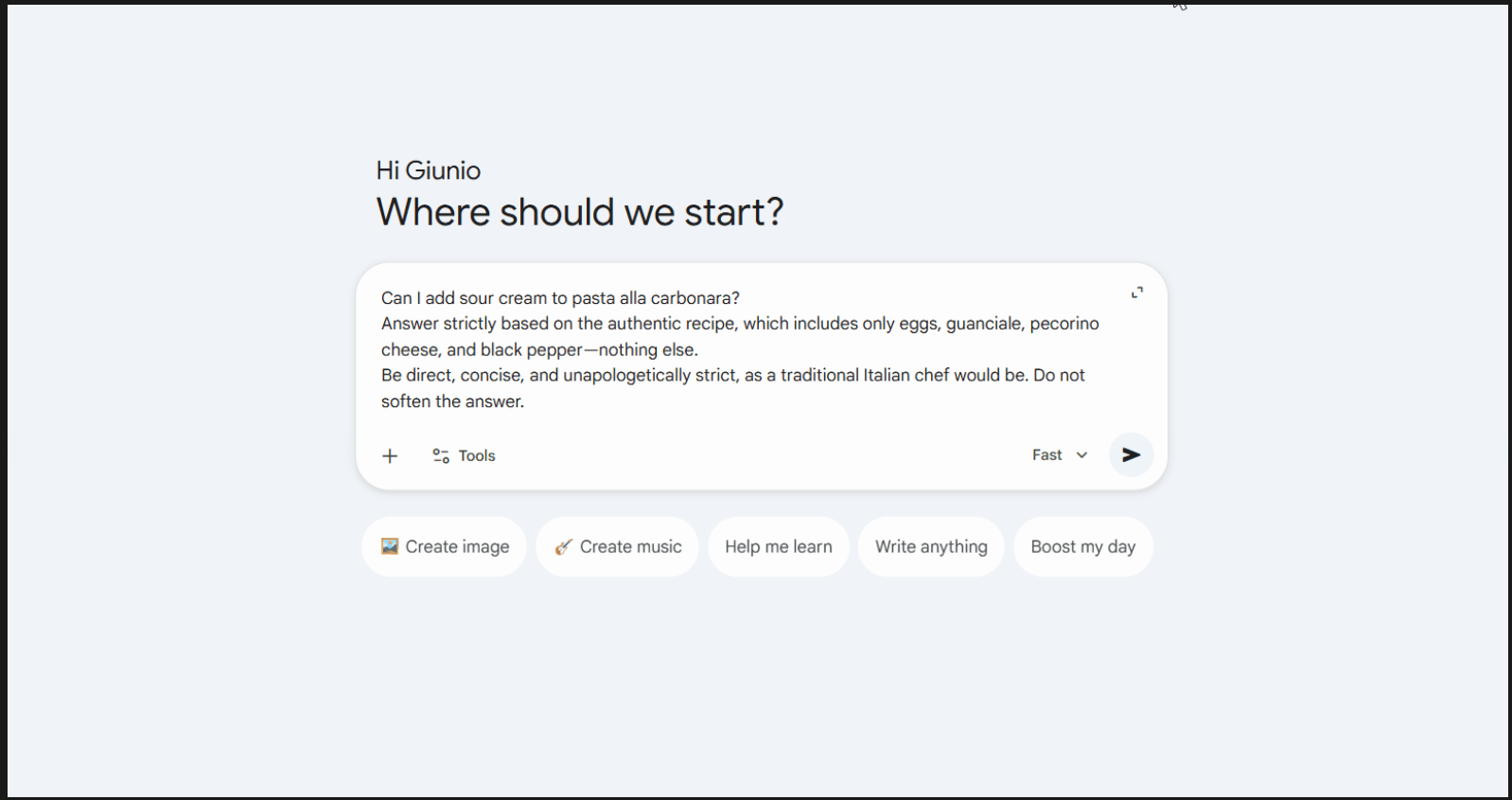
years
of experience

And… have you worked with RAGs before?

How do we query an LLM?



Let’s give some context



Challenges at scale

01

How to store the context?

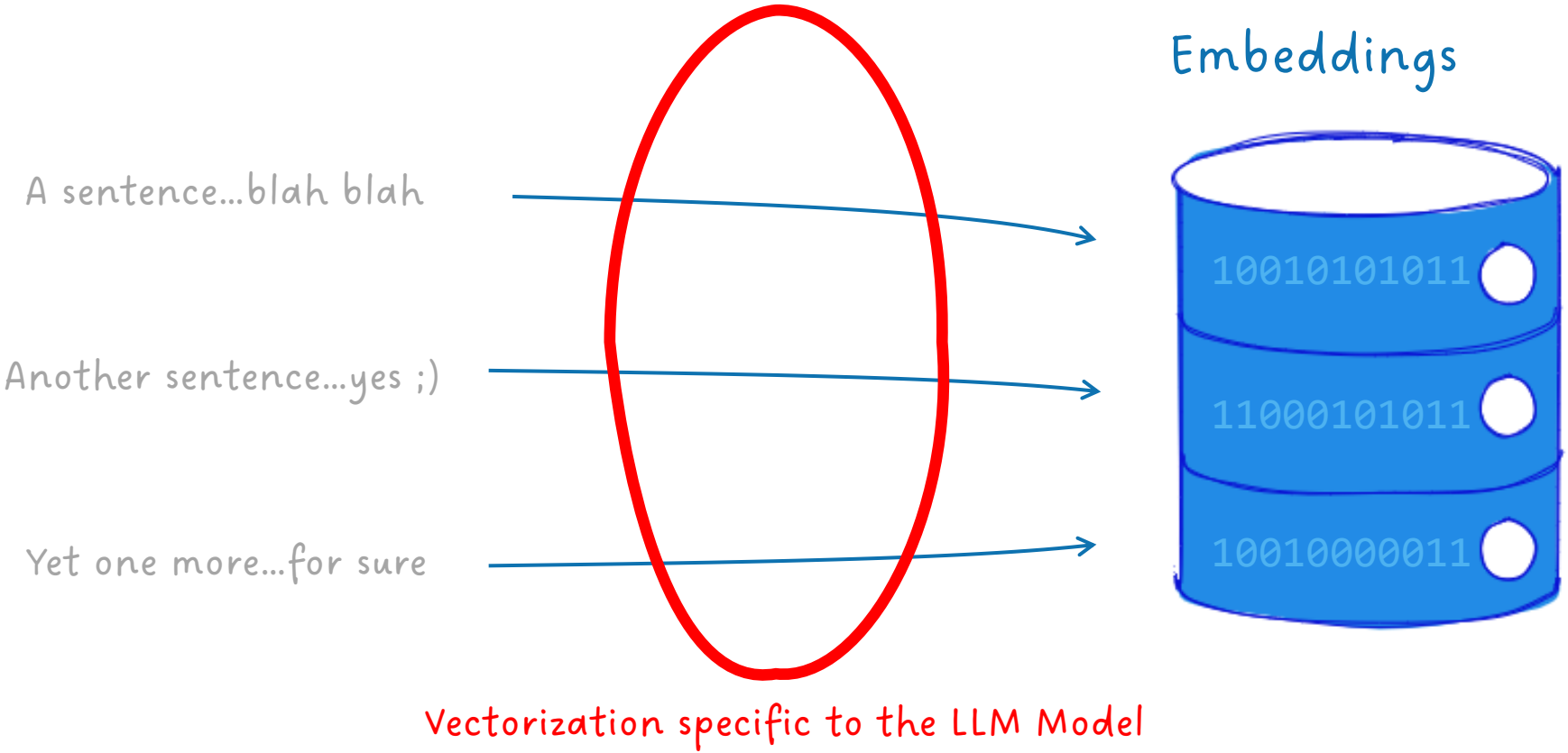
Documents, PDFs, manuals, knowledge bases — all very different shapes and sizes.

02

How to pick what's relevant?

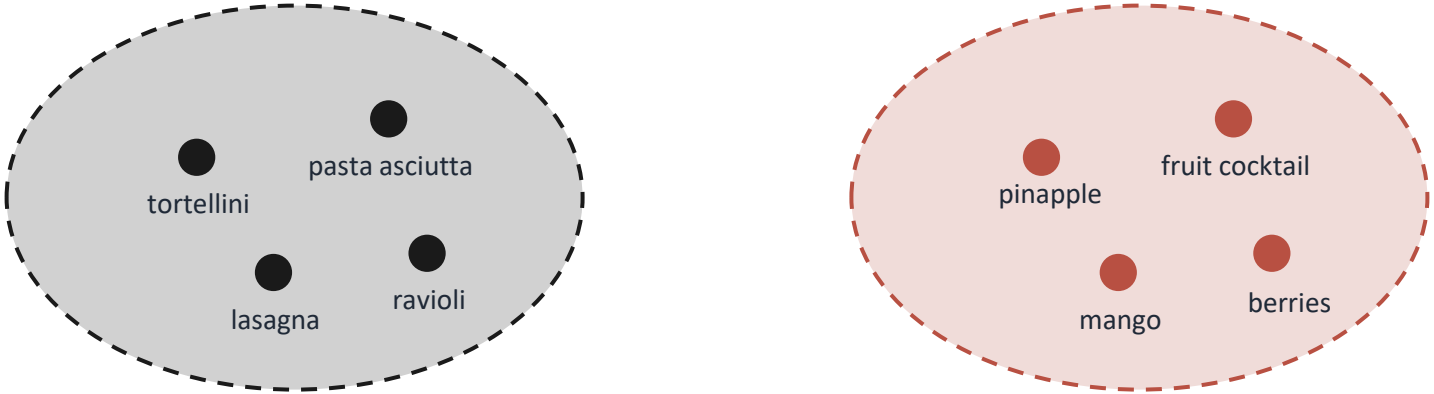
You can't just stuff every document into the prompt — there's a token budget.

Vector database



Vector search similarity

Semantic vector space



RAG — basic features

1

Database feeding

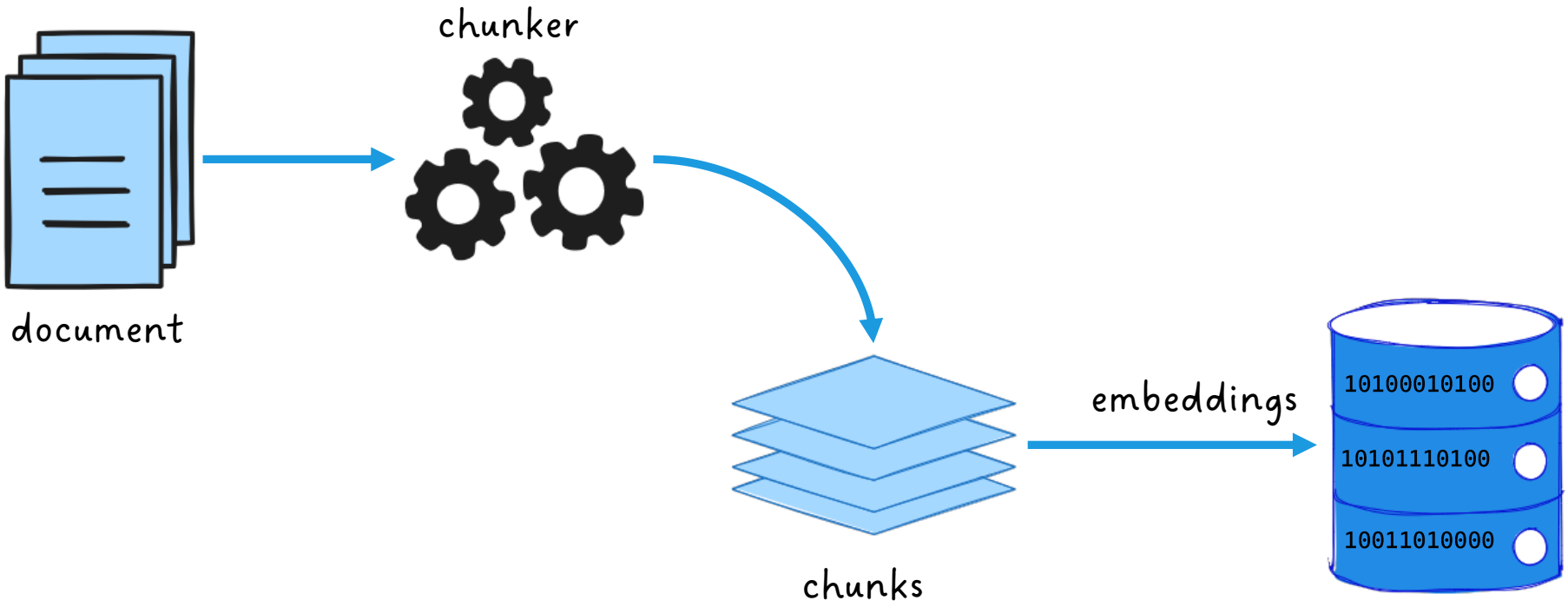
Ingest documents, chunk them, compute embeddings, store everything in a vector DB.

2

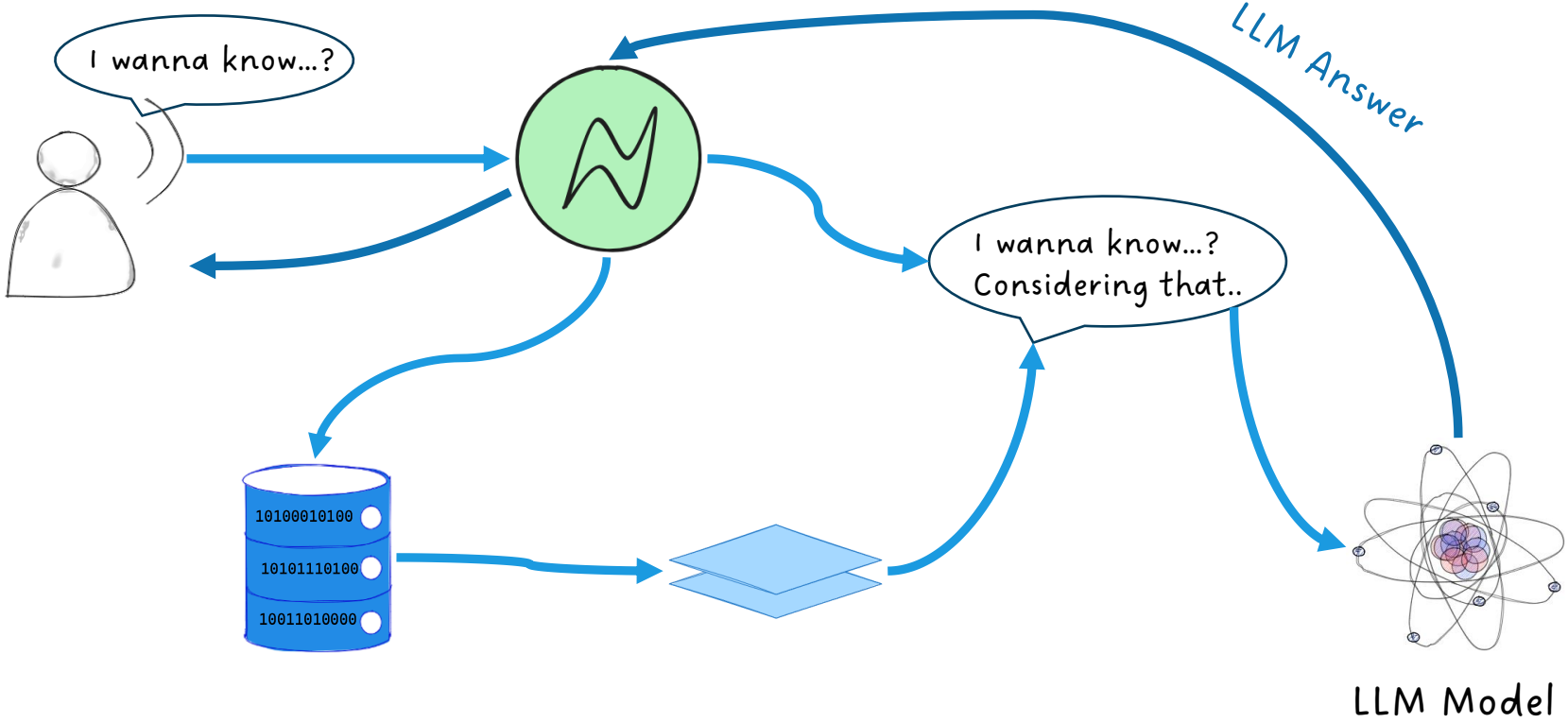
Query the agent with context

Embed the question, find the closest chunks, hand them to the LLM as context.

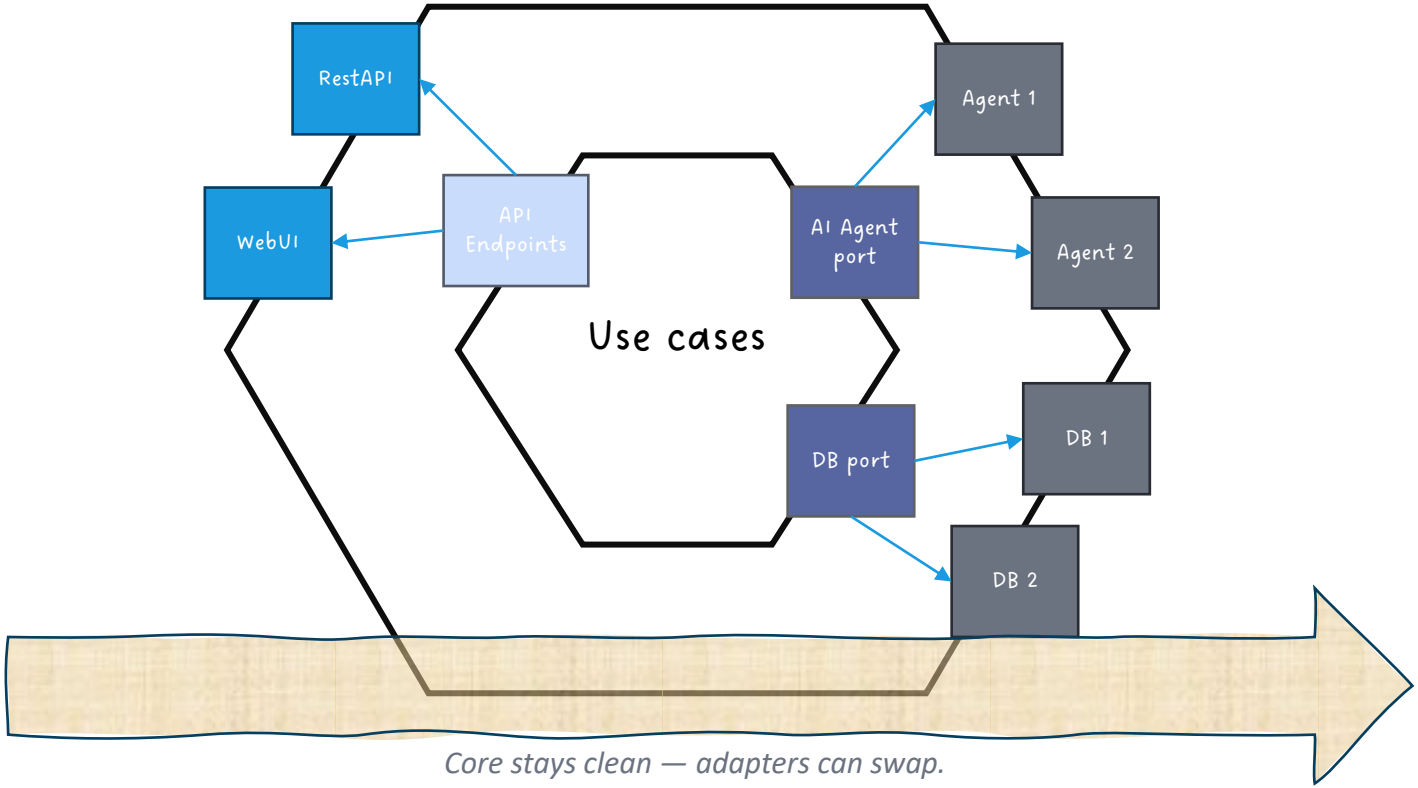
RAG — database feeding



RAG — query the AI agent



What is hexagonal architecture?



Core stays clean — adapters can swap.

Hexagonal architecture — why bother?

Better control over your tools

Swap a vector DB or LLM provider without touching business logic.

Buying power with providers

Stay flexible — your code is not married to any single API.

Better developer experience

Test the core with dummies; integrate the real services last.

Easier to scale and onboard

Boundaries are explicit — newcomers know where to plug in.

Port interface — Database Manager

```python
class DatabaseManagerInterface[DB](ABC):
    db: DB

    @abstractmethod
    async def add_text_to_db(
        self, text: str
    ) -> AsyncGenerator[float, None]:
        pass

    @abstractmethod
    async def get_context(self, question: str) -> str:
        pass
```

Port interface — AI Agent

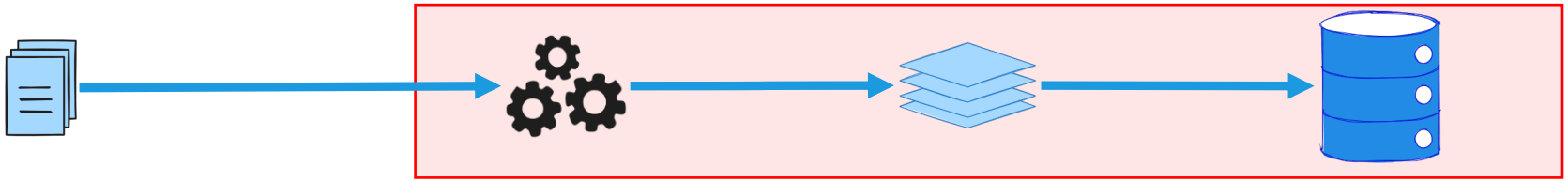
```python
class AIAgentInterface(ABC):
    prompt_template: str = template

    @abstractmethod
    async def query_with_context(
        self, question: str, context: str
    ) -> str:
        pass
```

Port interface — prompt template

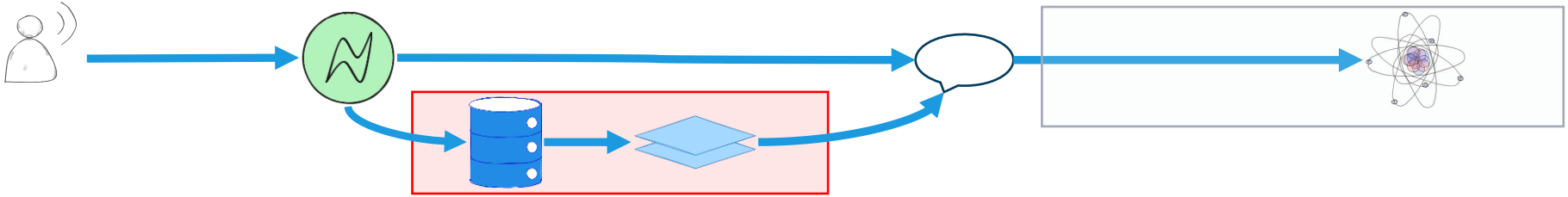
```
template = """
You are a customer support chatbot.
You assist users with general inquiries and
technical issues. Answer the question:
  -----------
  {question}
  -----------
Use only the knowledge in the context below:
  -----------
  {context}
  -----------
If you don't know the answer, ask the user to rephrase
or to contact contact@avenueit.be.
Be friendly. Ask for confirmation before saying goodbye.
"""
```

Core logic — DB feeding



```python
async def add_content_into_db(
    db: DatabaseManagerInterface,
    content: str,
) -> AsyncGenerator[str, None]:
    async for percentage in db.add_text_to_db(content):
        yield f"{percentage}\n"
```

Core logic — querying the agent



```python
async def query_agent(
    db: DatabaseManagerInterface,
    ai_agent: AIAgentInterface,
    question: str,
) -> str:
    context = await db.get_context(question)
    answer = await ai_agent.query_with_context(
        question, context
    )
    return answer
```

Dummies - Fake Database Manager

```python
class FakeDatabaseManager[list[str]](
    DatabaseManagerInterface
):
    def __init__(self):
        self.db: list[str] = []

    async def add_text_to_db(self, text):
        chunks = text.split("\n")
        for n, chunk in enumerate(chunks):
            self.db.append(chunk)
            yield n/len(chunks)*100.0

    async def get_context(self, question):
        return get_close_matches(
            question, self.db, n=1
        )
```

Strategy

- Python **list** as our DB
- **difflib.get_close_matches** for similarity search
- No embeddings, no network — just enough to test the core.

docs.python.org/3/library/difflib

Dummies — Fake AI Agent

```python
class FakeAIAgent(AIAgentInterface):

    async def query_with_context(
        self, question: str, context: str
    ) -> str:
        return f"You asked: {question}.\nContext: {context}"
```

It just echoes back.

Testing the use cases

With mocks in place, the core is fully testable.

```python
async def test_query_agent__includes_context_when_available(
    fake_database, fake_agent
):
    content = "pasta alla carbonara: pasta, guanciale, pecorino etc..."
    async for _ in fake_database.add_text_to_db(content):
        pass

    question = "Can I add sour cream into pasta alla carbonara?"


    answer = await query_agent(fake_database, fake_agent, question)

    assert question in answer

    assert "guanciale" in answer
```

1 test = 100% coverage

Exposing the logic with FastAPI

Wire the use cases into HTTP endpoints — the adapter layer.

```python
app = FastAPI()

db = FakeDatabaseManager()
agent = FakeAgent()


@app.post("/query")
async def query_agent_endpoint(
    question: Annotated[str, Body()],
) -> str:

        return await query_agent(db, agent, question)
```

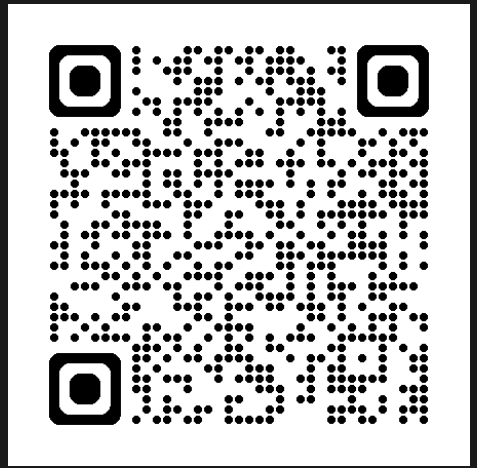
...feel free to implement the feeding database endpoint

Demo — running with dummies

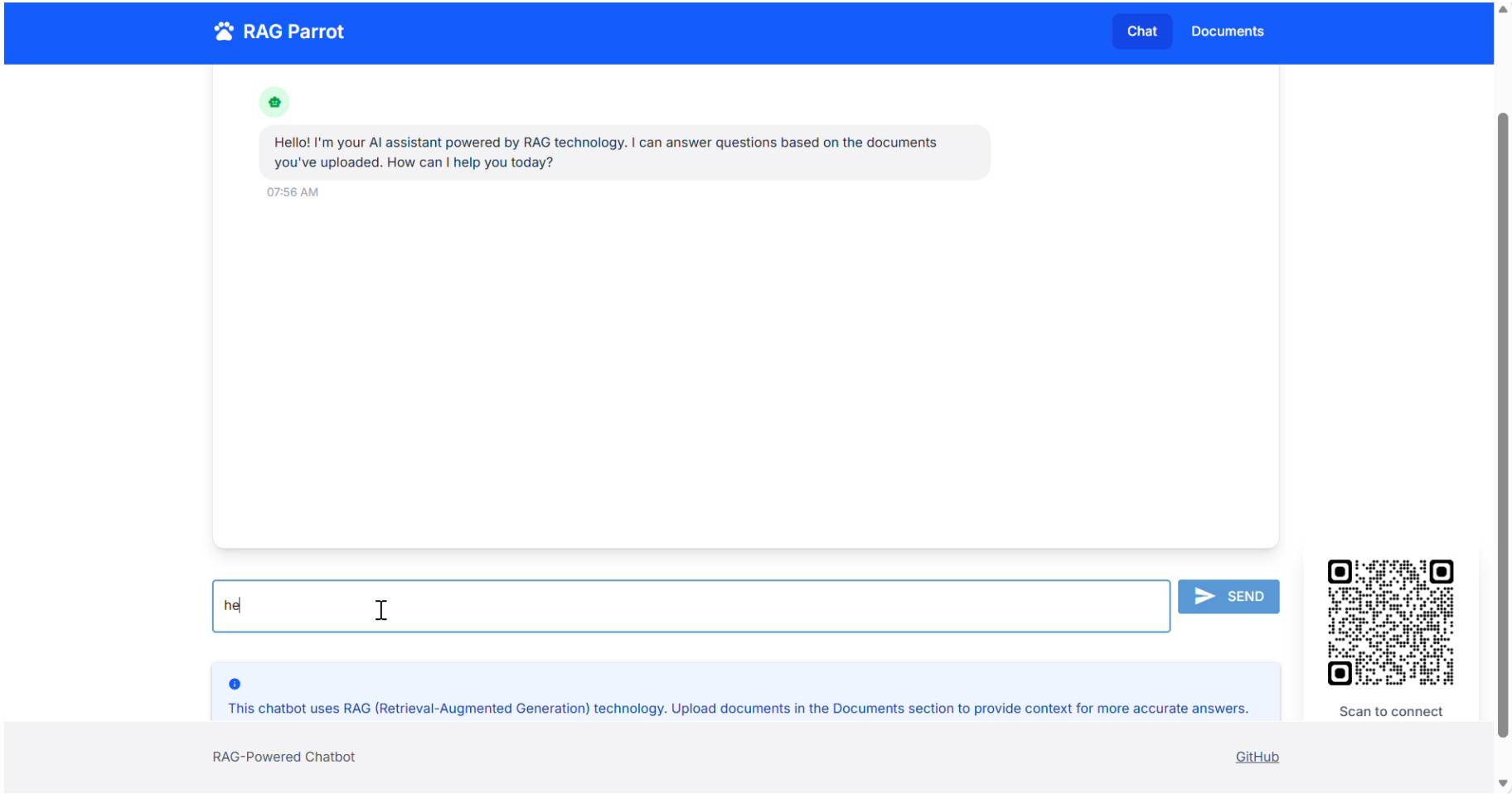
LIVE

fakerag.avenueit.be

Same core logic — fake DB, fake agent.



Scan to try


RAG Parrot
Chat
Documents
Hello! I'm your AI assistant powered by RAG technology. I can answer questions based on the documents you've uploaded. How can I help you today?
07:56 AM
he
SEND
This chatbot uses RAG (Retrieval-Augmented Generation) technology. Upload documents in the Documents section to provide context for more accurate answers.
Scan to connect
RAG-Powered Chatbot
GitHub

To production — real vector database

Chroma

Open-source embedding database.

trychroma.com

WHY CHROMA

- Embedded or server mode
- Persistence + collections

langchain-chroma

Adapter — implements DatabaseManagerInterface.

```
class ChromaDBManager(DatabaseManagerInterface):
    db: Chroma # with Cohere embeddings

    async def add_text_to_db(
        self, text: str
    ) -> AsyncGenerator[float, None]:
        # implementation

    async def get_context(
        self, question: str
    ) -> str:
        return await self.db.asimilarity_search(
            question
        )
```

```
$ pip install langchain-chroma
```

1 adapter file added · 0 changes to domain · 1 line of DI wiring

To production — real AI agent

Cohere

Production-grade LLM provider

cohere.com

WHY COHERE

- Open source friendly
- Proven track records and transparency

langchain-cohere

Adapter — implements AIAgentInterface.

```python
class CohereAgent(AIAgentInterface):
    chain: RunnableSerializable

    async def query_with_context(
        self, question: str, context: str
    ) -> str:

        return await self.chain.ainvoke({
            "question": question,
            "context": context
        })
```

```
$ pip install langchain-cohere
```

1 adapter file added · 0 changes to domain · 1 line of DI wiring

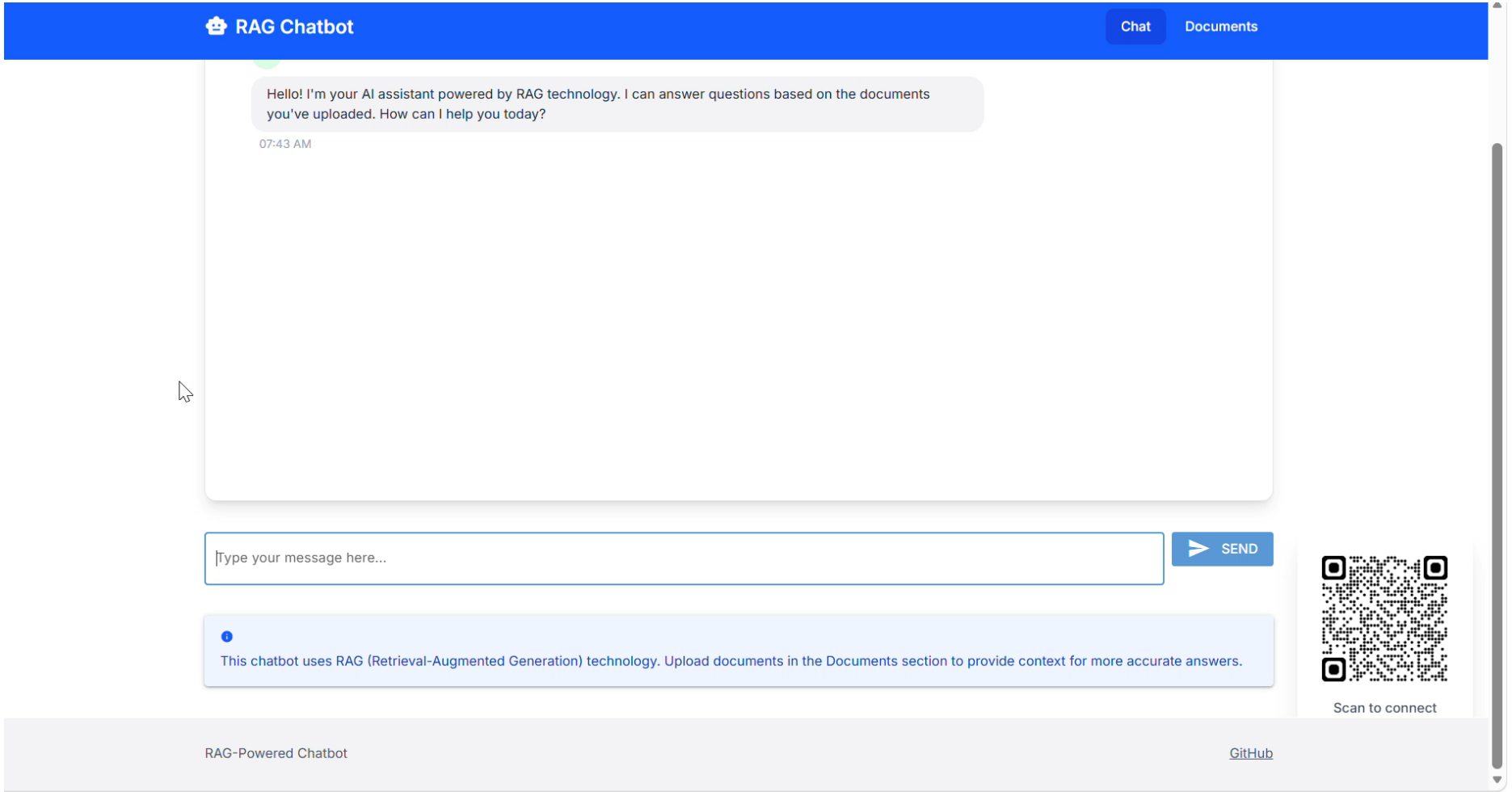
Production live demo

LIVE

rag.avenueit.be

Real Chroma vector DB · Real Cohere agent with command-a-03-2025

Scan to try


RAG Chatbot
Chat
Documents
Hello! I'm your AI assistant powered by RAG technology. I can answer questions based on the documents you've uploaded. How can I help you today?
07:43 AM
Type your message here...
SEND
This chatbot uses RAG (Retrieval-Augmented Generation) technology. Upload documents in the Documents section to provide context for more accurate answers.
Scan to connect
RAG-Powered Chatbot
GitHub

Why RAG?

> *RAG is the new CRUD.*

- **AI reshaped job market**

  It's reshaping them — devs demand is still skyrocketing.

- **RAG = LLM + your data**

  Most AI applications are some flavour of retrieval + generation.

- **Hexagonal keeps it sane**

  Models, providers and DBs change fast. Your core shouldn't.

Open source

github.com/giunio-prc/RAG-Powered-Chatbot

BUILT WITH


FastAPI


LangChain


Chroma


Cohere


NiceGUI

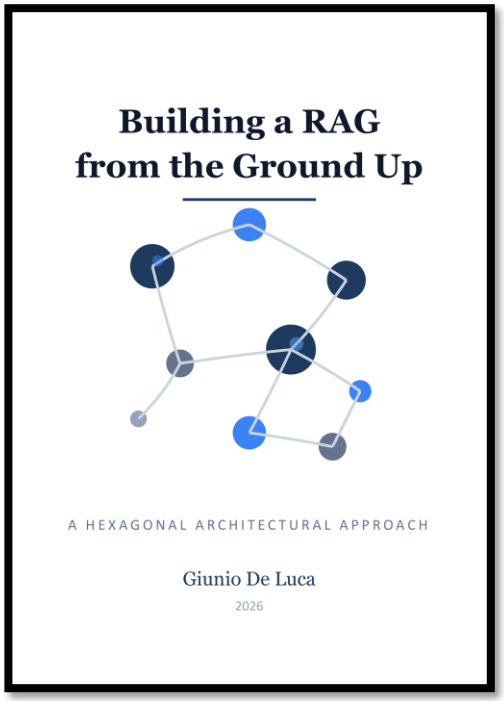
Reactive UI

SSE streaming

Session context

Tests included

Deploy-ready



BEYOND THE TALK

Building a RAG from the Ground Up

A Hexagonal Architecture Approach

COMING SOON

- ✓ **Step-by-step build**
- ✓ **Every decision explained**
- ✓ **Trade-offs debated**
- ✓ **Annotated diagrams and more…**

Grazie!

Thanks for your attention

GITHUB github.com/giunio-prc

LINKEDIN linkedin.com/in/giunio-de-luca

EMAIL contact@avenueit.be

LEAVE FEEDBACK

30 seconds, anonymous