Day 8: Database Implementation – NoSQL & Vector Databases

> Constraint respected: **No Docker is used** anywhere in this day's implementation.

This day focuses on **document databases, vector databases, and cloud object storage**, which together form the backbone of modern **AI-driven backend systems**.

---

1. MongoDB – Document Model & Validation

MongoDB Document Model

- Schema-flexible (but schema-controlled via validation)
- JSON-like documents (BSON)
- Optimized for nested, evolving data

Database & Collection Setup

```
use ai_app
```

Collection with Validation Rules

```
db.createCollection("users", {
  validator: {
    $jsonSchema: {
      bsonType: "object",
      required: ["name", "email", "created_at"],
      properties: {
        name: {
          bsonType: "string",
          description: "must be a string and is required"
        },
        email: {
          bsonType: "string",
          pattern: "^.+@.+$",
          description: "must be a valid email"
        },
        created_at: {
          bsonType: "date"
        }
      }
```

```
    }
  }
})
```

Insert & Query

```
db.users.insertOne({
  name: "Ashutosh",
  email: "ashu@example.com",
  created_at: new Date()
})

// Query
db.users.find({ email: "ashu@example.com" })
```

✔️Checklist: MongoDB collections with validation rules

---

2. Vector Databases – Conceptual Overview

Why Vector Databases

- Traditional DBs fail at **semantic similarity**
- Vectors encode meaning, not keywords

Use cases: - Semantic search - Recommendation engines - RAG (Retrieval Augmented Generation) - Chat memory & AI assistants

---

3. ChromaDB (Local, No Docker)

Installation

```
pip install chromadb sentence-transformers
```

Initialize ChromaDB

```
import chromadb
from chromadb.config import Settings

client = chromadb.Client(
    Settings(persist_directory="./chroma_data")
)
```

```
collection = client.get_or_create_collection(name="docs")
```

Insert Sample Data

```
documents = [
    "PostgreSQL is a relational database",
    "MongoDB is a document-oriented NoSQL database",
    "Vector databases enable semantic search",
]

ids = ["doc1", "doc2", "doc3"]

collection.add(documents=documents, ids=ids)
```

✔️Checklist: Vector database configured with sample data

---

4. Embedding Generation Pipeline

SentenceTransformer-Based Embeddings

```
from sentence_transformers import SentenceTransformer

model = SentenceTransformer("all-MiniLM-L6-v2")

def generate_embedding(text: str):
    return model.encode(text).tolist()
```

Store with Embeddings

```
embeddings = [generate_embedding(d) for d in documents]

collection.add(
    documents=documents,
    embeddings=embeddings,
    ids=ids
)
```

✔️Checklist: Embedding generation pipeline implemented

---

5. Semantic Search with Similarity Threshold

Query with Threshold

```
query = "Which database is NoSQL?"
query_embedding = generate_embedding(query)

results = collection.query(
    query_embeddings=[query_embedding],
    n_results=3
)

for doc, score in zip(results['documents'][0], results['distances'][0]):
    if score < 0.4:  # similarity threshold
        print("MATCH:", doc, "score:", score)
```

Similarity Notes

- Lower distance = higher similarity
- Threshold tuning is **domain-specific**

✔️Checklist: Semantic search with threshold working

---

6. Weaviate (Conceptual + Optional Local Setup)

When to Use Weaviate

- Hybrid search (vector + filters)
- Large-scale production systems
- Multi-tenant vector workloads

(ChromaDB is enough for local & interview-level mastery)

---

7. Cloud Object Storage – S3 & GCS

Why Object Storage

- Store embeddings metadata, PDFs, images
- Cheap, scalable, durable

---

8. AWS S3 Integration (Signed URLs)

Install SDK

```
pip install boto3
```

Generate Signed Upload URL

```
import boto3

s3 = boto3.client("s3")

url = s3.generate_presigned_url(
    ClientMethod="put_object",
    Params={
        "Bucket": "my-ai-bucket",
        "Key": "uploads/file.txt"
    },
    ExpiresIn=3600
)

print(url)
```

✔️Checklist: S3 signed URL integration complete

---

9. GCP Cloud Storage (Signed URLs)

Install SDK

```
pip install google-cloud-storage
```

Signed URL Example

```
from google.cloud import storage

client = storage.Client()
bucket = client.bucket("my-gcs-bucket")
blob = bucket.blob("uploads/file.txt")

url = blob.generate_signed_url(expiration=3600, method="PUT")
print(url)
```

✔️Checklist: GCS signed URL integration complete

Day 8 Completion Status

- ✅MongoDB collections with validation rules
- ✅Vector DB (ChromaDB) configured & tested
- ✅Embedding generation pipeline implemented
- ✅Semantic search with similarity threshold
- ✅AWS S3 & GCP signed URL integration

Architectural Insight (Important)

Typical AI backend stack:

```
Client → API (FastAPI)
      → MongoDB (metadata)
      → Vector DB (semantic retrieval)
      → LLM
      → S3 / GCS (documents, files)
```

Next logical progression: **Day 9 – Backend Performance & Scaling** (Caching, async IO, batching, N+1 problems, load testing)

Or we can **merge Days 6–8 into a single AI-ready backend architecture**.