**Phase 2 – MVP Bot (Week 3–4): Implementation Guide**

This pack gives you ready-to-run scaffolding for:

* **Rooms skill** with SQL filters (city, budget, occupancy) → top‑k results
* **FAQ skill (RAG v1)** using Qdrant + sentence-transformers → cited answers
* **LangGraph** orchestration with simple Router → Rooms | FAQ | Fallback
* **FastAPI** /chat endpoint that executes the graph and returns JSON
* **Pytest** golden‑path tests (rooms, faq, unknown, error)

Assumptions: You finished Phase 1 (Docker Compose skeleton, Postgres + Qdrant running, basic data loaded).

**1) Project Layout**

chatbi/

app/

\_\_init\_\_.py

config.py

db.py

models.py

repositories/

rooms\_repo.py

faq\_repo.py

rag/

embed.py

index\_faqs.py

retriever.py

reranker.py (optional later)

graph/

state.py

router.py

nodes\_rooms.py

nodes\_faq.py

nodes\_fallback.py

graph.py

api/

server.py

schemas.py

utils/pii.py (phase 3)

tests/

test\_rooms.py

test\_faq.py

test\_unknown.py

test\_error.py

data/

hotels.csv

room\_rates.csv

faqs.jsonl

docker/

docker-compose.yml

.env

pyproject.toml

README.md

**2) Config + Dependencies**

**pyproject.toml (snippet)**

[project]

name = "chatbi"

version = "0.2.0"

dependencies = [

"fastapi",

"uvicorn[standard]",

"pydantic>=2",

"sqlalchemy",

"sqlmodel",

"psycopg[binary]",

"qdrant-client",

"sentence-transformers",

"langgraph",

"langchain-text-splitters",

"tenacity",

"python-dotenv",

"pytest",

]

**.env (example)**

POSTGRES\_URL=postgresql+psycopg://postgres:postgres@localhost:5432/chatbi

QDRANT\_URL=http://localhost:6333

EMBED\_MODEL=sentence-transformers/all-MiniLM-L6-v2

TOPK=5

**docker/docker-compose.yml (snippet)**

services:

app:

build: ..

env\_file: ../.env

depends\_on: [postgres, qdrant]

ports: ["8000:8000"]

postgres:

image: postgres:16

environment:

POSTGRES\_PASSWORD: postgres

POSTGRES\_DB: chatbi

ports: ["5432:5432"]

qdrant:

image: qdrant/qdrant:latest

ports: ["6333:6333"]

volumes:

- qdrant\_storage:/qdrant/storage

volumes:

qdrant\_storage:

**3) Database Layer**

**app/models.py**

from sqlmodel import SQLModel, Field

from typing import Optional

class Hotel(SQLModel, table=True):

hotel\_id: int | None = Field(default=None, primary\_key=True)

name: str

city: str

country: str

stars: Optional[int] = None

lat: Optional[float] = None

lon: Optional[float] = None

amenities\_json: Optional[str] = None

class RoomRate(SQLModel, table=True):

id: int | None = Field(default=None, primary\_key=True)

hotel\_id: int = Field(foreign\_key="hotel.hotel\_id")

room\_type: str

occupancy: int

currency: str

base\_rate: float

refundable: bool

breakfast\_included: bool

class Policy(SQLModel, table=True):

id: int | None = Field(default=None, primary\_key=True)

hotel\_id: int = Field(foreign\_key="hotel.hotel\_id")

key: str

value: str

**app/db.py**

import os

from sqlmodel import SQLModel, create\_engine, Session

engine = create\_engine(os.getenv("POSTGRES\_URL"), pool\_pre\_ping=True)

def init\_db():

SQLModel.metadata.create\_all(engine)

def get\_session():

return Session(engine)

**app/repositories/rooms\_repo.py**

from typing import List, Optional

from sqlmodel import select

from ..db import get\_session

from ..models import RoomRate, Hotel

class RoomsRepo:

def \_\_init\_\_(self):

self.session = get\_session()

def search(

self,

city: str,

max\_price: Optional[float] = None,

occupancy: Optional[int] = None,

topk: int = 5,

) -> List[dict]:

q = (

select(RoomRate, Hotel)

.join(Hotel, Hotel.hotel\_id == RoomRate.hotel\_id)

.where(Hotel.city.ilike(city))

)

if max\_price is not None:

q = q.where(RoomRate.base\_rate <= max\_price)

if occupancy is not None:

q = q.where(RoomRate.occupancy >= occupancy)

q = q.limit(topk)

rows = self.session.exec(q).all()

results = []

for rr, h in rows:

results.append({

"hotel": h.name,

"city": h.city,

"room\_type": rr.room\_type,

"occupancy": rr.occupancy,

"price": rr.base\_rate,

"currency": rr.currency,

"refundable": rr.refundable,

"breakfast\_included": rr.breakfast\_included,

})

return results

**4) FAQ RAG v1 (Embeddings + Qdrant)**

**app/rag/embed.py**

import os

from sentence\_transformers import SentenceTransformer

\_model = None

def get\_embedder():

global \_model

if \_model is None:

\_model = SentenceTransformer(os.getenv("EMBED\_MODEL", "sentence-transformers/all-MiniLM-L6-v2"))

return \_model

def embed\_texts(texts: list[str]) -> list[list[float]]:

model = get\_embedder()

return model.encode(texts, normalize\_embeddings=True).tolist()

**app/rag/index\_faqs.py**

import json

import os

from qdrant\_client import QdrantClient

from qdrant\_client.http.models import Distance, VectorParams, PointStruct

from .embed import embed\_texts

COLL = "faqs"

def ensure\_collection(client: QdrantClient, dim: int):

client.recreate\_collection(

collection\_name=COLL,

vectors\_config=VectorParams(size=dim, distance=Distance.COSINE),

)

def run\_index(path="data/faqs.jsonl"):

client = QdrantClient(url=os.getenv("QDRANT\_URL", "http://localhost:6333"))

questions, payloads = [], []

with open(path, "r", encoding="utf-8") as f:

for line in f:

obj = json.loads(line)

q = obj["question"].strip()

a = obj["answer"].strip()

payloads.append({"question": q, "answer": a, "hotel\_id": obj.get("hotel\_id"), "city": obj.get("city"), "lang": obj.get("lang", "en")})

questions.append(q)

vectors = embed\_texts(questions)

ensure\_collection(client, len(vectors[0]))

points = [PointStruct(id=i, vector=vectors[i], payload=payloads[i]) for i in range(len(vectors))]

client.upsert(collection\_name=COLL, points=points)

return len(points)

if \_\_name\_\_ == "\_\_main\_\_":

n = run\_index()

print(f"Indexed {n} FAQ entries")

**app/rag/retriever.py**

import os

from qdrant\_client import QdrantClient

from .embed import embed\_texts

COLL = "faqs"

def retrieve(query: str, topk: int = 5):

client = QdrantClient(url=os.getenv("QDRANT\_URL", "http://localhost:6333"))

vec = embed\_texts([query])[0]

res = client.search(collection\_name=COLL, query\_vector=vec, limit=topk)

hits = []

for p in res:

payload = p.payload

hits.append({

"question": payload.get("question"),

"answer": payload.get("answer"),

"score": float(p.score),

"meta": {k: payload.get(k) for k in ("hotel\_id", "city", "lang")},

})

return hits

**5) LangGraph Orchestration**

**app/graph/state.py**

from pydantic import BaseModel

from typing import Optional, List, Literal

class GraphState(BaseModel):

user\_text: str

intent: Optional[Literal["rooms", "faq", "unknown"]] = None

city: Optional[str] = None

budget: Optional[float] = None

occupancy: Optional[int] = None

answer: Optional[str] = None

results: Optional[list] = None

citations: Optional[List[dict]] = None

**app/graph/router.py**

import re

from .state import GraphState

ROOMS\_KW = re.compile(r"\b(room|rooms|book|price|rate)\b", re.I)

FAQ\_KW = re.compile(r"\b(check[- ]?in|policy|refund|breakfast|parking|wifi|faq|question)\b", re.I)

CITY\_RE = re.compile(r"in\s+([A-Za-z][A-Za-z\s]+)$", re.I)

PRICE\_RE = re.compile(r"under\s+(\d+(?:\.\d+)?)|max\s+(\d+(?:\.\d+)?)", re.I)

OCC\_RE = re.compile(r"for\s+(\d+)\s\*(?:people|guests)?", re.I)

def router\_node(state: GraphState) -> GraphState:

text = state.user\_text.strip()

if ROOMS\_KW.search(text):

state.intent = "rooms"

elif FAQ\_KW.search(text):

state.intent = "faq"

else:

state.intent = "unknown"

# naive slot extraction

if m := CITY\_RE.search(text):

state.city = m.group(1).strip()

if m := PRICE\_RE.search(text):

state.budget = float(next(g for g in m.groups() if g))

if m := OCC\_RE.search(text):

state.occupancy = int(m.group(1))

return state

**app/graph/nodes\_rooms.py**

from .state import GraphState

from ..repositories.rooms\_repo import RoomsRepo

repo = RoomsRepo()

def rooms\_node(state: GraphState) -> GraphState:

if not state.city:

state.answer = "Please tell me the city to search rooms."

return state

results = repo.search(city=state.city, max\_price=state.budget, occupancy=state.occupancy, topk=5)

state.results = results

if results:

bullets = [f"{r['hotel']} – {r['room\_type']} (occ {r['occupancy']}) {r['currency']} {r['price']:.2f}" for r in results]

state.answer = "Here are some options:\n" + "\n".join(bullets)

else:

state.answer = "I couldn't find rooms matching your filters. Try adjusting budget or occupancy."

return state

**app/graph/nodes\_faq.py**

from .state import GraphState

from ..rag.retriever import retrieve

def faq\_node(state: GraphState) -> GraphState:

hits = retrieve(state.user\_text, topk=5)

state.citations = hits

if hits:

best = hits[0]

state.answer = f"{best['answer']}\n\nTop related questions:\n" + "\n".join([f"- {h['question']} (score {h['score']:.2f})" for h in hits[:3]])

else:

state.answer = "I couldn't find an answer in FAQs."

return state

**app/graph/nodes\_fallback.py**

from .state import GraphState

def fallback\_node(state: GraphState) -> GraphState:

state.answer = "Sorry, I didn't understand. You can ask about rooms or hotel policies."

return state

**app/graph/graph.py**

from langgraph.graph import StateGraph, END

from .state import GraphState

from .router import router\_node

from .nodes\_rooms import rooms\_node

from .nodes\_faq import faq\_node

from .nodes\_fallback import fallback\_node

def build\_graph():

sg = StateGraph(GraphState)

sg.add\_node("router", router\_node)

sg.add\_node("rooms", rooms\_node)

sg.add\_node("faq", faq\_node)

sg.add\_node("fallback", fallback\_node)

def route\_edge(s: GraphState):

return s.intent or "unknown"

sg.add\_edge("router", "rooms", condition=lambda s: route\_edge(s) == "rooms")

sg.add\_edge("router", "faq", condition=lambda s: route\_edge(s) == "faq")

sg.add\_edge("router", "fallback", condition=lambda s: route\_edge(s) == "unknown")

sg.set\_entry\_point("router")

sg.add\_edge("rooms", END)

sg.add\_edge("faq", END)

sg.add\_edge("fallback", END)

return sg.compile()

**6) FastAPI /chat Endpoint**

**app/api/server.py**

from fastapi import FastAPI

from pydantic import BaseModel

from ..graph.graph import build\_graph

from ..graph.state import GraphState

from ..db import init\_db

app = FastAPI(title="Chatbi")

workflow = build\_graph()

class ChatRequest(BaseModel):

message: str

class ChatResponse(BaseModel):

reply: str

intent: str | None = None

results: list | None = None

citations: list | None = None

@app.on\_event("startup")

def on\_start():

init\_db()

@app.post("/chat", response\_model=ChatResponse)

def chat(req: ChatRequest):

state = GraphState(user\_text=req.message)

out = workflow.invoke(state)

return ChatResponse(

reply=out.answer or "",

intent=out.intent,

results=out.results,

citations=out.citations,

)

**Run**

uvicorn chatbi.app.api.server:app --reload --port 8000

**7) Golden-Path Tests (pytest)**

**tests/test\_rooms.py**

import requests

def test\_rooms\_paris():

r = requests.post("http://localhost:8000/chat", json={"message": "show me rooms in Paris under 200 for 2"})

j = r.json()

assert j["intent"] == "rooms"

assert j["results"] is not None

**tests/test\_faq.py**

import requests

def test\_faq\_checkin():

r = requests.post("http://localhost:8000/chat", json={"message": "what is the check-in time?"})

j = r.json()

assert j["intent"] == "faq"

assert j["citations"] is not None

assert len(j["citations"]) > 0

**tests/test\_unknown.py**

import requests

def test\_unknown():

r = requests.post("http://localhost:8000/chat", json={"message": "tell me a joke"})

j = r.json()

assert j["intent"] == "unknown"

assert "rooms" in j["reply"].lower()

**tests/test\_error.py**

import requests

def test\_error\_handles\_missing\_city():

r = requests.post("http://localhost:8000/chat", json={"message": "show me rooms"})

j = r.json()

assert j["intent"] == "rooms"

assert "city" in j["reply"].lower()

**8) Data Prep Quickstart**

Load example rows into Postgres (replace with your real data from Phase 1).

INSERT INTO hotel (hotel\_id, name, city, country, stars) VALUES

(1, 'Chatbi Paris', 'Paris', 'France', 4),

(2, 'Chatbi Rome', 'Rome', 'Italy', 4)

ON CONFLICT DO NOTHING;

INSERT INTO roomrate (hotel\_id, room\_type, occupancy, currency, base\_rate, refundable, breakfast\_included) VALUES

(1, 'Standard Room', 2, 'EUR', 145, true, false),

(1, 'Deluxe', 2, 'EUR', 220, true, true),

(2, 'Standard', 2, 'EUR', 130, false, false)

ON CONFLICT DO NOTHING;

Create data/faqs.jsonl:

{"hotel\_id":1,"city":"Paris","lang":"en","question":"What time is check-in?","answer":"Check-in starts at 3 PM."}

{"hotel\_id":1,"city":"Paris","lang":"en","question":"Is breakfast included?","answer":"Breakfast is included only for Deluxe rooms."}

{"hotel\_id":2,"city":"Rome","lang":"en","question":"Do you have free Wi‑Fi?","answer":"Yes, complimentary Wi‑Fi is available in all rooms."}

Index FAQs:

python -m chatbi.app.rag.index\_faqs data/faqs.jsonl

**9) cURL Smoke Test**

curl -s -X POST http://localhost:8000/chat \

-H 'Content-Type: application/json' \

-d '{"message":"show me rooms in Paris under 200 for 2"}' | jq

Expected (example):

{

"reply": "Here are some options:\nChatbi Paris – Standard Room (occ 2) EUR 145.00",

"intent": "rooms",

"results": [ {"hotel":"Chatbi Paris", ...} ],

"citations": null

}

**10) What to Tweak Next (still Phase 2 scope)**

* Improve router regexes (add more cities/phrases).
* Accept explicit structured payloads: {city,budget,occupancy} when available.
* Add limit param to /chat → pass into RoomsRepo.search.
* Normalize currencies (EUR/USD) if dataset mixes.
* Return citations for rooms (e.g., SQL trace or hotel\_id) for debugging.

**11) Ready‑Made Task List (copy/paste into issues)**

1. Wire app/db.init\_db() into startup and confirm tables exist.
2. Load seed Hotel and RoomRate records.
3. Create data/faqs.jsonl; run index\_faqs.py.
4. Launch uvicorn and hit /chat with rooms + faq prompts.
5. Add pytest tests and make them pass.
6. Add Docker Compose service for app and confirm cross‑container connectivity.

**12) Troubleshooting**

* **qdrant\_client.exceptions.UnexpectedResponse** → collection not created; run index\_faqs.py once.
* **No rooms returned** → check city casing; .ilike(city) expects string like "Paris"; ensure seed data.
* **Embedding download slow** → pre‑download model by running embed.py once in the container.
* **CORS** from frontend → add fastapi.middleware.cors (if needed later).

You now have all core pieces for Phase 2. Next phase will add memory, guardrails, and observability (Redis, Presidio, Langfuse).