Routing Contract v1

Routing Contract v1

This is the foundation. We will not change behavior later unless the doc forces us to.

Purpose (why this exists)

The document requires:

- A **Router Node**
- **Conditional Routing**
- Automatic **tool selection**
- Priority on **correct routing over answer quality**

This contract defines **deterministic rules** so the agent behaves predictably.

Conversation Memory

- **Last 10 messages only**
- Passed unchanged through all nodes
  (Required by “Maintain conversation state and multi-turn history”)

Primary Routes

1. Postgres Route

When to use
User asks about:

- Customers
- Tickets
- Accounts
- Status of an issue
- Customer-specific information

Examples

- “Show tickets for customer John”
- “What is the status of ticket 12?”
- “Which city is customer 3 from?”

Data allowed

- `customers`
- `tickets`
- Simple joins only

If tool fails OR returns empty
➡️ Fallback to **LLM Response**

📌 Source:

- *Conditional Routing*: “Customer/order/account info → Postgres”
- *PostgreSQL section* (defined tables)

---

2. Vector DB Route

When to use
User asks about:

- FAQs
- Help articles
- Support documentation
- “How do I…?” product/support questions

Examples

- “How do I reset my password?”
- “What is your refund policy?”
- “Explain ticket escalation”

Behavior

- Semantic search only
- Top relevant chunks returned

If tool fails OR returns empty
➡️ Fallback to **LLM Response**

📌 Source:

- *Conditional Routing*: “Knowledge-base/FAQ/support article → Vector DB”
- *Vector Database section*

---

3. External Tool Route (Mock API)

When to use
User asks for:

- General information
- Data NOT in Postgres or Vector DB

Examples

- “What’s the weather today?”
- “What is the price of Bitcoin?”
- “Give me today’s temperature in London”

Behavior

- Return predefined mock responses only

If tool fails OR returns empty
➡️ Fallback to **LLM Response**

📌 Source:

- *Conditional Routing*: “General information → External API Tool”
- *Tools section* (mock external API)

---

4. LLM Default Route

When to use

- Query does not clearly match any route
- All tools fail or return empty
- Conversational or clarifying questions

Examples

- “Can you help me?”
- “What can you do?”

- “Explain this system”

📌 Source:

- *LLM Response Node*
- *Fallback implied by routing design*

Fallback Rules (STRICT)

Fallback happens when:

- Tool raises an error **OR**
- Tool returns no meaningful data

Fallback target:
➡️ **LLM Response Node**

(No retry loops. No second tool attempts.)

Routing Priority Order

1. Postgres
2. Vector DB
3. External Tool
4. LLM Default

(The first confident match wins.)

Tool Interface Contracts

Tool Interface Contracts (schemas only, no implementation)

This step is required because the document mandates **Structured Tool Calling** and automatic selection .
We define **what tools accept and return** so routing + agent logic stays clean.

---

1. Postgres Tool (SQL)

Purpose (doc-backed):
Retrieve customer and ticket data from PostgreSQL

Input schema

- `query` (string) — **read-only SQL**
- `params` (optional object) — named parameters

Output schema

- `rows` (array of objects)
- `row_count` (number)

Constraints

- SELECT only
- Tables limited to `customers`, `tickets`
- Empty result → fallback to LLM

---

2. Vector DB Tool (Semantic Search)

Purpose (doc-backed):
Search FAQs / support articles from vector database

Input schema

- `query` (string)

- `top_k` (number, default 3)

Output schema

- `documents` (array)
  - `content` (string)
  - `score` (number)

Constraints

- Search-only
- No uploads
- Empty result → fallback to LLM

---

3. External Tool (Mock API)

Purpose (doc-backed):
Simulate external APIs like weather or crypto

Input schema

- `type` (enum: `weather`, `crypto`)
- `query` (string)

Output schema

- `result` (string)
- `source` (string: `"mock"`)

Constraints

- Hardcoded responses
- Failure or unknown query → fallback to LLM

Why we stop here

- Agent + LangGraph depends on **stable tool contracts**
- Prevents rework
- Matches “Correct routing > answer quality”

Router Node Specification

Router Node Specification (no code)

This step is **explicitly required** by the document:

- “Create a graph with: **Router Node**”
- “Conditional Routing”

We now define **how the router thinks**, not how it’s coded.

Router Node — Purpose

Given:

- User message
- Last **10 messages** of conversation

The router must:

1. Select **one primary route**
2. Allow **fallback** if the tool fails or returns empty

No tool execution happens here.

Routes the Router Can Output

The router must return **exactly one** of:

- `POSTGRES`
- `VECTOR`
- `EXTERNAL`
- `LLM`

(These map 1:1 to nodes in LangGraph.)

Routing Rules (deterministic)

Route = `POSTGRES`

If the user asks about:

- Customers
- Tickets
- Account-related status
- Customer-specific data

Signal words / intent

- customer, ticket, issue, status, account, id, city

Route = `VECTOR`

If the user asks about:

- How-to
- Policies
- FAQs
- Support explanations

Signal words / intent

- how do I, help, policy, guide, explain, support

Route = `EXTERNAL`

If the user asks about:

- General real-world info
- Data not stored internally

Signal words / intent

- weather, temperature, price, crypto, bitcoin

Route = `LLM`

If:

- The query is conversational
- No route matches confidently
- The user is asking about the system itself

Fallback Logic (graph-level)

Fallback is **not decided by the router**.

Fallback happens when:

- Tool throws an error **OR**
- Tool returns empty data

Then:
➡️ Route automatically to **LLM Response Node**

(This satisfies “Allow fallback” without complicating the router.)

Router Prompt (conceptual, not code)

> “Classify the user’s intent into exactly one route: POSTGRES, VECTOR, EXTERNAL, or LLM.
> Choose POSTGRES for customer or ticket data.
> Choose VECTOR for FAQ or support knowledge.
> Choose EXTERNAL for general real-world information.
> Choose LLM if none apply.
> Return only the route name.”

Why this is intentionally strict

- Prevents multi-route confusion
- Easy to test
- Matches document’s conditional routing requirement
- Keeps agent behavior predictable

LangGraph Layout

LangGraph Layout (nodes & edges only)

This step maps **directly** to the document requirement:

> “Create a graph with: Router Node, Vector Search Node, Postgres Query Node, External Tool Node, LLM Response Node”

No code. Just structure.

Nodes (exactly 5)

1. **Router Node**
   - Input: user message + last 10 messages
   - Output: one route (`POSTGRES | VECTOR | EXTERNAL | LLM`)
2. **Postgres Query Node**
   - Calls Postgres Tool
   - Returns rows or empty/error
3. **Vector Search Node**
   - Calls Vector DB Tool
   - Returns documents or empty/error
4. **External Tool Node**
   - Calls Mock External Tool
   - Returns result or empty/error
5. **LLM Response Node**
   - Produces final user-facing answer
   - Used for:
     - Direct LLM route
     - All fallbacks

Graph Flow (simple & deterministic)

Start → Router Node

From **Router Node**:

- `POSTGRES` → Postgres Query Node
- `VECTOR` → Vector Search Node

- `EXTERNAL` → External Tool Node
- `LLM` → LLM Response Node

From **Tool Nodes**:

- If **success with data** → LLM Response Node
- If **error OR empty** → LLM Response Node (fallback)

End → LLM Response Node

Key Design Decisions (document-aligned)

- No loops
- No retries
- No multi-tool chaining
- One router decision per user turn

This keeps:

- Routing correct
- Debugging easy
- Behavior explainable

Why this satisfies the document

- Agentic graph ✅
- Conditional routing ✅
- Tool-based execution ✅
- Fallback handled cleanly ✅

Minimal Implementation Order

Minimal Implementation Order (what to code, in what order)

This step exists to **avoid over-engineering** and to stay aligned with the document's scope .

Implementation Order (strict)

1️⃣ Tools first (no agent yet)

Why:
The document requires **structured tool calling**.
If tools don't work alone, the agent will fail silently.

Code in this order:

1. Postgres Tool
    - Connect
    - Run SELECT
    - Return rows / empty
2. Vector Tool
    - Load dummy docs
    - Search
    - Return matches
3. External Tool
    - Hardcoded responses

➡️ Each tool must be callable directly and tested alone.

📌 Source: *Tools (Structured Tool Calling)*

2️⃣ Router Node (standalone)

Why:
Routing correctness is your top priority.

What we implement:

- Router prompt

- Route output validation
- Unit test with example queries

📌 Source: *Router Node + Conditional Routing*

3️⃣ LangGraph wiring

Why:
Once tools + router work, wiring is mechanical.

What we add:

- Nodes
- Conditional edges
- Fallback to LLM

📌 Source: *Agentic AI (LangChain + LangGraph)*

4️⃣ FastAPI endpoint

Why:
Thin transport layer only.

What we expose:

- `POST /chat`
- Streaming response

📌 Source: *FastAPI Backend*

5️⃣ Streamlit UI (last)

Why:
UI should not influence agent design.

What we build:

- Simple chat UI
- Token streaming

📌 Source: *Streamlit Frontend*

What we intentionally delay

- UI polish
- Advanced memory
- Dynamic tool discovery
- Uploading vector data

All outside document scope.

Define Fake Data & Dummy Content

Define Fake Data & Dummy Content (exact, minimal)

This step is required so tools can be **tested independently**, as implied by:

- PostgreSQL section (defined tables)
- Vector Database section (pre-load 2–3 articles)

No flexibility here — we define **exact data**.

1️⃣ PostgreSQL Fake Data

Tables (exactly as document states)

📌 Source: *PostgreSQL section*

customers

id	name	city
1	John Doe	New York
2	Jane Smith	London
3	Alex Brown	Toronto

tickets

id	customer_id	issue	status
1	1	Login not working	open
2	1	Password reset issue	closed
3	2	Billing question	open

Why this is enough

- Covers single-table queries
- Covers joins
- Covers empty-result cases

2️⃣ Vector Database Dummy Articles

📌 Source: *Vector Database section*

We preload **exactly 3 articles**:

Article 1

Title: Password Reset Guide
Content:
“How to reset your password: click ‘Forgot Password’, enter your email, and follow the reset link.”

Article 2

Title: Ticket Escalation Policy
Content:
“Tickets are escalated if unresolved for more than 48 hours. Escalated tickets are reviewed by senior support staff.”

Article 3

Title: Refund Policy
Content:
“Refunds are available within 14 days of purchase if the service was not used.”

Why only 3

- Document says *2–3 support articles*
- Enough to validate semantic routing
- No noise

3️⃣ External Tool Mock Responses

📌 Source: *External API Tool (mock)*

- Weather →
  “The weather today is sunny with a temperature of 25°C.”
- Crypto →
  “Bitcoin price is $30,000.”

Unknown queries → empty → fallback.

We stop here intentionally

At this point:

- Tool behavior is fully spec’d
- Test cases are obvious
- No agent code yet

Implement Tools Only

Implement Tools Only (still no agent, no graph)

This step is justified by the document's **Structured Tool Calling** requirement and the defined Postgres / Vector / External tools .
We implement **just enough code** to prove tools work independently.

What we will implement (exactly)

1️⃣ Postgres Tool

Does

- Connect to Postgres
- Run **SELECT-only** queries
- Return `{ rows, row_count }`

Does NOT

- Write data
- Auto-generate SQL
- Handle retries

Success criteria

- Can fetch:
  - All customers
  - Tickets by customer
  - Join customers + tickets
- Empty result returns cleanly (no exception)

2️⃣ Vector Search Tool

Does

- Load the 3 predefined articles
- Embed once at startup

- Run semantic search
- Return top matches with scores

Does NOT

- Accept uploads
- Re-embed dynamically

Success criteria

- “How do I reset my password?” → Password article
- Irrelevant query → empty result

3️⃣ External Mock Tool

Does

- Return hardcoded responses for:
  - weather
  - crypto
- Unknown query → empty result

Does NOT

- Call real APIs

File structure (minimal)

```
tools/
  postgres_tool.py
  vector_tool.py
  external_tool.py
```

Each tool:

- Callable directly
- Returns plain JSON-like dicts
- No LangChain wrappers yet

What we explicitly delay

- LangChain `Tool` objects
- LangGraph
- FastAPI
- Streaming
- UI

All intentionally postponed.