**🔷 Why This Project?**

In many large organizations, the **Service Catalog is considered the “Bible” for the DBG (Digital Business Group)** and other development teams. It contains vital metadata about APIs, microservices, owners, systems, and flows. It is **frequently used by developers, architects, support teams, and product owners** for day-to-day work.

However, despite its importance, accessing the right information is time-consuming due to the following challenges:

**🔧 Common Pain Points**

* 🔍 **Information is fragmented** — spread across webpages, Word documents, Confluence, Excel files, and emails.
* 🧠 **Tribal knowledge dependency** — only certain individuals “know where to look”.
* 🧾 **Lack of standardized search** — users rely on CTRL+F, folder digging, or team Slack messages.
* ⏱️ **Wasted effort** — gathering service info (contact, flow, diagram, backend) can take 30+ minutes.
* 🧩 **Unclear ownership** — developers don’t always know who owns a specific service or its contact point.
* 🔄 **Repeated queries** — same information is asked repeatedly across different teams.
* ❌ **No easy onboarding** — new team members struggle to navigate the service ecosystem quickly.

**💡 Use Case**

We propose building a **Service Catalog Copilot** — a smart assistant that understands natural language questions and retrieves structured and unstructured service data from your internal sources.

Instead of searching through multiple folders, docs, and spreadsheets, users can simply ask:

* “Who owns the payment-service?”
* “Show APIs related to eligibility”
* “What does auth-service do?”
* “Where is the architecture diagram for drug pricing?”

**✅ Solution Overview**

* A **FastAPI-based backend** that routes requests
* **LangChain-powered RAG** pipeline to combine document and DB-based answers
* **Azure OpenAI** to understand natural language
* **FAISS vector store** to search unstructured files like .docx, .txt, and .pdf
* **PostgreSQL database** to store metadata (service name, API name, contact, diagram links)
* A clean **Chat UI** (web-based interface) to ask questions and view responses in a user-friendly format
* Deployed securely on Azure with monitoring and token usage tracking

**🏆 Benefits**

| **Feature** | **Benefit** |
| --- | --- |
| 🔍 Natural Language Interface | Ask questions instead of searching manually |
| ⏱️ Fast Access to Knowledge | Save 30+ minutes per query |
| 🧠 Knowledge Centralization | Reduces dependency on individuals |
| 🚀 Faster Onboarding | New hires get context instantly |
| 📉 Lower SME Load | Reduce repetitive questions to leads and architects |
| 🌐 Scalable Architecture | Securely hosted on Azure with cloud-native stack |

**🧰 Technology Stack**

| **Component** | **Tool** |
| --- | --- |
| Backend | FastAPI (Python) |
| LLM Engine | Azure OpenAI (GPT-3.5/4 via LangChain) |
| Semantic Search | FAISS Vector Store (HuggingFace Embeddings) |
| Structured DB | PostgreSQL |
| File Support | Word, Text, PDF |
| Frontend | Web Chat UI (React/Tailwind or Streamlit optional) |
| Hosting | Azure App Service or AKS |
| Monitoring | Azure Monitor, App Insights |

**👨‍💻 Required Skill Set**

| **Role** | **Skills Needed** |
| --- | --- |
| Backend Developer | FastAPI, Pydantic, LangChain integration |
| AI Engineer | Prompt engineering, RAG pipelines, vector DBs |
| Chat UI Developer | React or Streamlit, REST API integration |
| DevOps Engineer | Azure App Service, secure deployment, CI/CD |
| Analyst | Document tagging, catalog cleanup, embeddings prep |

**🗓️ Timeline to Deliver (MVP)**

| **Phase** | **Duration** | **Deliverables** |
| --- | --- | --- |
| 🔍 Requirement Analysis | 3–5 days | Use case walkthrough, DB schema review |
| 🧱 Design | 4–6 days | LangChain + RAG prompt flow, architecture design |
| 🛠️ Development | 10–14 days | Backend API, Vector DB, UI integration |
| 🧪 Testing | 4–5 days | Functional + user testing of queries |
| 🚀 Deployment & Rollout | 3–4 days | Azure deployment, CI/CD setup, stakeholder demo |

**⏳ Total Duration: ~1 month for MVP delivery**

