CodeWells: Agentic RAG Chatbot for Developers

Objective

To build an *Agentic Retrieval-Augmented Generation (RAG) Chatbot* that assists developers and team members by providing intelligent, context-aware help during development blockers. The chatbot should access **Confluence pages**, **GitHub repositories**, and **Q&A documentation**, with **agentic reasoning** to take context-driven actions.

Phase 1: Requirement Gathering & Architecture Design

Tasks:

1. Identify the primary developer pain points (types of blockers).
2. Define chatbot scope — internal code help, documentation queries, debugging tips, etc.
3. Decide on communication channel (e.g., web UI via Streamlit).
4. Design high-level system architecture: RAG pipeline + Agent layer + Integration layer.
5. Define data flow: query → retrieval → reasoning → response/action.
6. Select models and embedding strategies (e.g., text-embedding-3-small).
7. Prepare documentation of architecture and dependencies.

Phase 2: Data Integration & Preparation

Tasks:

1. Integrate **Confluence API** to fetch internal pages (REST API).
2. Connect to **GitHub API** for code and repository access.

3. Gather **internal Q&A and wiki documentation**.
4. Implement data cleaning, metadata tagging (source, timestamp, author).
5. Chunk and embed documents using sentence transformers or OpenAI embeddings.
6. Store embeddings in a **Vector Database (FAISS / Chroma)**.
7. Schedule periodic data sync jobs.

Phase 3: RAG Pipeline Development

Tasks:

1. Implement retrieval logic for top-k relevant documents.
2. Use LangChain or Google ADK for context-aware RAG flow.
3. Design **context builder** to merge retrieved data with user query.
4. Integrate reasoning layer — the **Agent** that decides:
    - Whether to fetch data, generate answers, or call an API (GitHub/Confluence).
5. Optimize prompt templates for developer queries.
6. Test RAG responses for correctness, context quality, and hallucination rate.

Phase 4: Agentic Layer & Orchestration

Tasks:

1. Implement **Google ADK Agents** with task orchestration capabilities.
2. Define agent roles:
    - **Research Agent** → fetches relevant documents.
    - **Explain Agent** → summarizes or explains code/docs.

    - **Action Agent** → performs tasks like fetching pull requests or creating issues.
3. Add **Human-in-the-Loop (HIL)** for critical actions (e.g., PR creation, code merge).
4. Enable **tool calling** for actions (GitHub API, Confluence write).
5. Maintain agent memory (conversation and context).
6. Conduct dry runs and refine agentic workflows.

Phase 5: Backend & Frontend Integration

Tasks:

1. Create **FastAPI backend** for orchestrating chat requests and model calls.
2. Build **Streamlit UI** for conversational interface:
    - Real-time chat window
    - Code/Markdown rendering
    - Source citation display
3. Implement authentication (Google OAuth / company SSO).
4. Integrate logging and telemetry (query latency, response quality).
5. Establish async communication between Streamlit ↔ FastAPI.

Phase 6: Testing & Evaluation

Tasks:

1. Unit test each module — data retrieval, agent reasoning, response generation.
2. Perform **end-to-end user scenario tests** (e.g., “Show me last merged PR”, “Explain code in repo X”).

3. Conduct internal user testing with developers.
4. Evaluate accuracy, relevance, latency, and user satisfaction metrics.
5. Fix edge cases (e.g., missing data, incorrect references).

Phase 7: Deployment & Documentation

Tasks:

1. Containerize app using **Docker**.
2. Deploy on **GCP / internal Kubernetes cluster**.
3. Set up CI/CD pipeline for automatic deployment.
4. Create developer onboarding guide and usage manual.
5. Prepare presentation & POC demonstration script.
6. Collect post-deployment feedback and improvement suggestions.