Project Documentation

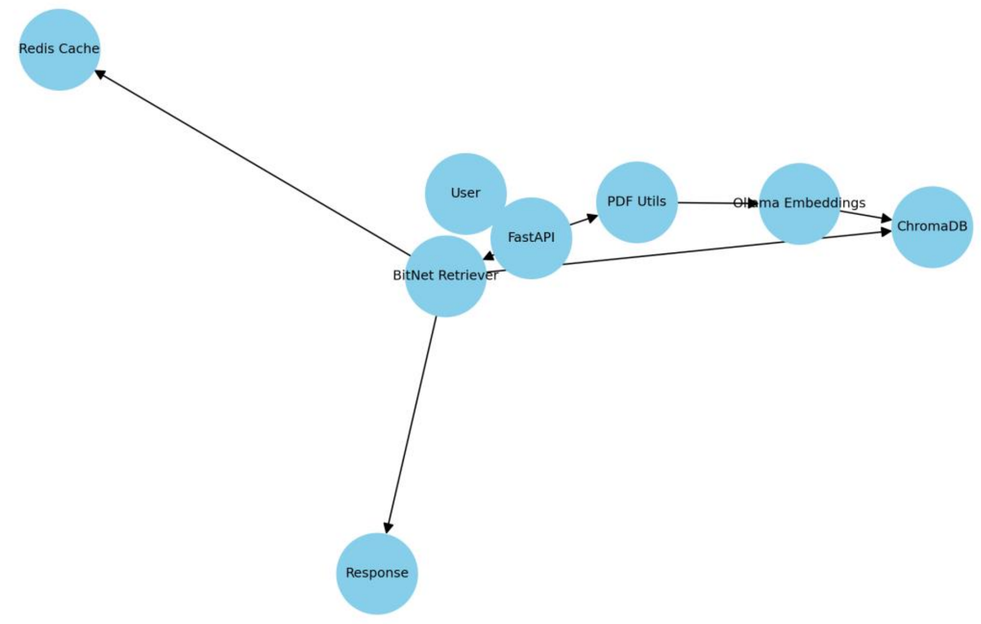
1. Project Overview

This project is a Document Question Answering (DocQA) system. It allows users to upload PDFs, which are processed into embeddings for semantic search and question answering. The system integrates FastAPI (backend APIs), ChromaDB (vector database), Redis (caching), Ollama (for embeddings), and BitNet (for retrieval).

2. Architecture & Flow

The system workflow is as follows:
1. User uploads a PDF through FastAPI.
2. PDF text is extracted using PDF utilities.
3. Extracted text is chunked and embedded using Ollama Embeddings API.
4. Embeddings are stored in ChromaDB.
5. Redis caches frequent query results.
6. When a query is submitted, the retriever (BitNet) fetches relevant chunks from ChromaDB and generates an answer.

System Architecture Diagram:



3. Directory Structure

```
├── app.py              # FastAPI entry point
├── settings.py         # Project configuration file
├── utils/
│   ├── chroma_utils.py    # ChromaDB helper functions
│   ├── pdf_utils.py       # PDF text extraction helpers
│   ├── redis_utils.py     # Redis cache management
├── create_env.sh        # Virtual environment setup script
├── requirements.txt      # Python dependencies
├── logs/               # Log files directory
└── systemd/             # Service management files
```

4. Key Modules Explained

- app.py: Defines FastAPI routes `/upload-pdf` and `/query`.
- chroma_utils.py: Functions to insert/query vectors in ChromaDB.
- pdf_utils.py: Extracts and preprocesses text from PDFs.
- redis_utils.py: Implements caching for faster query performance.
- settings.py: Central configuration (DB URLs, API endpoints, logging).

5. Embedding & Retriever Models

Embeddings Model (Ollama):
- Used for generating vector embeddings from text.
- Common models include Phi-3, Mistral, and Gemma (lightweight LLMs).
- These embeddings are stored in ChromaDB for semantic similarity search.

Retriever Model (BitNet):
- Lightweight LLM retriever designed for RAG (Retrieval Augmented Generation).
- Takes user query, retrieves top-k embeddings from ChromaDB, and returns relevant chunks.
- Produces context-aware answers by combining query and retrieved context.

6. Setup & Deployment

1. Run `bash create_env.sh` to set up the Python virtual environment.
2. Install dependencies: `pip install -r requirements.txt`.
3. Update `settings.py` with correct DB URLs and API endpoints.
4. Start app locally: `uvicorn app:app --reload`.
5. For production, configure `systemd` service files.

7. API Endpoints

- POST /upload-pdf: Upload PDF and store embeddings.
  Request: Multipart PDF file
  Response: { 'message': 'PDF processed successfully' }

- POST /query: Query stored documents.
  Request: { 'query': 'your question here' }
  Response: { 'results': [...] }

8. Project Purpose & Use Cases

Purpose:
- To provide an intelligent document assistant that enables semantic search and Q&A over uploaded PDFs.
- Converts static documents into an interactive knowledge base.

Use Cases:
1. Enterprise Knowledge Base: Employees can search policies and manuals.
2. Healthcare: Doctors query patient guidelines and medical literature.
3. Legal: Lawyers search case law and judgments.
4. Customer Support: Agents access product documentation quickly.

9. Logging & Monitoring

- Logs are stored in the logs/ directory.
- Rotating logs configured for maintainability.
- Cron jobs can back up logs daily for auditing and debugging.

10. API cURL Examples

Here are sample cURL requests and responses for the available API endpoints:

1) Upload PDF

Request:
```bash
curl -X POST "https://api-auto-ticket-creation.c-zentrix.com/upload-pdf" \
  -H "accept: application/json" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@sample.pdf"
```

Response:

```json
{ "message": "PDF processed successfully" }
```

2) Query Documents

Request:
```bash
curl -X POST "https://api-auto-ticket-creation.c-zentrix.com/query" \
  -H "accept: application/json" \
  -H "Content-Type: application/json" \
  -d '{"query": "Explain the main concept from uploaded PDF"}'
```

Response:
```json
{ "results": ["Relevant passage from PDF"] }
```