RAG-Assisted Chatbot — Portfolio & Interview Assistant ■■

RAG-Assisted Chatbot is an installable Python package that powers a portfolio-driven HR interview assistant. It uses GitHub README scraping, PDF conversion, vectorization (ChromaDB), and Retrieval-Augmented Generation (RAG) combined with an LLM to answer HR-style questions based on a candidate's resume and repositories.

■ Highlights

- Scrapes GitHub repositories' README files and saves them as well-styled PDFs.
- Builds a persistent Chroma vector store from PDF content using SentenceTransformers embeddings.
- Queries the vector DB to fetch top-k relevant text chunks for a question (RAG).
- Uses an LLM (e.g., `gpt-5-mini` via `langchain-openai`) with structured output to produce interview-style responses.
- Packaged for easy reuse: import `Assistant`, `BuildVectorDB`, `AskToVectorDB`, and `GithubScrapper` from the package namespace. Example:

```
from rag_assisted_bot.rag_assisted_chatbot import Assistant, BuildVectorDB, AskToVectorDB, GithubScrapp
```

(When installed via pip the package name is `rag-assisted-chatbot`.)

Table of contents

■ Quick overview

This project provides three main capabilities:

- GitHub scraping: `GithubScrapper` fetches README content and converts to PDF (uses `xhtml2pdf`).
- Vector DB builder: `BuildVectorDB` loads PDFs, splits text into chunks, encodes with `sentence-transformers`, and stores embeddings in ChromaDB.
- Assistant: `Assistant` pairs an LLM with the RAG results to answer HR-style questions using an up-to-date resume context.

■ Installation

Clone and install locally:

```
git clone <repo_url>
cd <repo>
python -m pip install -e .
```

Or install dependencies from `requirements.txt`:

```
python -m pip install -r requirements.txt
```

> Tip: Use a virtual environment (venv/conda).

■■ Configuration

Project configuration values are in `rag_assisted_bot/rag_assisted_chatbot/config.py`:

- `GITHUB_USERNAME` — username to scrape (default: `"vijaytakbhate2002"`)
- `GITHUB_PDF_FOLDER` — folder to save generated PDFs (default: `"rag_assisted_bot/scrapped_data/github_pdfs"`)
- `METADATA_JSON_PATH` — metadata file location (default: `"scrapped_metadata/metadata.json"`)
- `VECTORDB_PATH` — path for persistent ChromaDB storage (default: `<project_root>/vectordb`)
- `COLLECTION_NAME` — Chroma collection name (default: `"my_embeddings"`)
- `EMBEDDING_MODEL_NAME` — embedding model (default: `"all-MiniLM-L6-v2"`)
- `TOP_K_MATCHES` — number of RAG results to include (default: 3)
- `GPT_MODEL_NAME` — model used by the assistant (default: `"gpt-5-mini"`)

Environment variables:

- `TOKEN_GITHUB` — **required** GitHub personal access token (the `GithubScrapper` will raise a `RuntimeError` if not set).

■ Quickstart Examples

1) Scrape READMEs and save PDFs

```
from rag_assisted_bot.rag_assisted_chatbot import GithubScrapper
import rag_assisted_bot.rag_assisted_chatbot.config as cfg

scraper = GithubScrapper(
   username=cfg.GITHUB_USERNAME,
   save_folder=cfg.GITHUB_PDF_FOLDER,
   metadata_save_folder=cfg.METADATA_JSON_PATH
)
scraper.scrap()
```

2) Build the persistent vector database

```
from rag_assisted_bot.rag_assisted_chatbot import BuildVectorDB
import rag_assisted_bot.rag_assisted_chatbot.config as cfg

builder = BuildVectorDB(
```

```
    directory_path=cfg.GITHUB_PDF_FOLDER,
    vectordb_path=cfg.VECTORDB_PATH,
    embedding_model_name=cfg.EMBEDDING_MODEL_NAME,
    collection_name=cfg.COLLECTION_NAME
)
builder.build(chunk_size=300, chunk_overlap=100)
```

3) Run the assistant (interactive programmatic use)

```
from rag_assisted_bot.rag_assisted_chatbot import Assistant
import rag_assisted_bot.rag_assisted_chatbot.config as cfg

assistant = Assistant(
    gpt_model_name=cfg.GPT_MODEL_NAME,
    temperature=0.7,
    collection_name=cfg.COLLECTION_NAME,
    vectordb_path=cfg.VECTORDB_PATH,
    rag_activated=True
)
result = assistant.chat_with_model("Tell me about your key MLOps achievements")
print(result['question_category'])
print(result['response'].model_dump())
```

4) Query the Vector DB directly

```
from rag_assisted_bot.rag_assisted_chatbot.ask_vectordb import AskToVectorDB
import chromadb
import rag_assisted_bot.rag_assisted_chatbot.config as cfg

client = chromadb.PersistentClient(path=cfg.VECTORDB_PATH)
collection = client.get_collection(name=cfg.COLLECTION_NAME)
asker = AskToVectorDB(collection=collection, embedding_model_name=cfg.EMBEDDING_MODEL_NAME)
res = asker.ask("Explain your role in the Medical Insurance project", n_results=3)
print(res['documents'][0])
```

■ How it works (architecture)

1. `GithubScrapper` fetches repository README files via GitHub API and saves them as PDFs.
2. `BuildVectorDB` loads PDFs with `PyMuPDF`, splits long texts into chunks using `RecursiveCharacterTextSplitter`, and encodes chunks with `SentenceTransformer`.
3. Embeddings are stored in a persistent ChromaDB collection (`my_embeddings`).
4. `AskToVectorDB` embeds queries and queries ChromaDB for top-k chunks.
5. `Assistant` obtains a category for the question, fetches RAG context (top-k chunks), and calls an LLM with a `SystemMessage` + retrieved context to provide a structured `InterViewResponse`.

Structured LLM output uses `pydantic` models from `rag_assisted_bot/rag_assisted_chatbot/output_structure.py` (fields include `response_message`, `reference_links`, `confidence_score`, `follow_up_question`).

■ Project structure

Key files and modules:

- `rag_assisted_bot/rag_assisted_chatbot/` — package modules
- `github_scrapper.py` — GitHub README → PDF (method: `scrap()`)
- `build_vectordb.py` — builds & persists ChromaDB embeddings (`BuildVectorDB`)
- `ask_vectordb.py` — queries vector DB (`AskToVectorDB`)
- `main.py` — `Assistant` class and RAG orchestration
- `prompts.py`, `references.py` — prompts and static resume content
- `output_structure.py` — pydantic models for structured outputs (`InterViewResponse`, `QuestionCategory`)
- `config.py` — defaults you can edit (paths & model names)
- Output folders created at runtime:
- `rag_assisted_bot/scrapped_data/` — PDFs (default: `rag_assisted_bot/scrapped_data/github_pdfs`)
- `scrapped_metadata/` — metadata JSON (default: `scrapped_metadata/metadata.json`)
- `vectordb/` (default persistent ChromaDB storage path)

■ Troubleshooting & tips

- Missing GitHub token: `GithubScrapper` will raise `RuntimeError` at import time if `TOKEN_GITHUB` is not set in the environment.
- If embeddings don't persist, ensure `VECTORDB_PATH` is writable and compatible with your `chromadb` version.
- PDF loading requires `PyMuPDF` (package name `PyMuPDF`) and the `langchain_community.document_loaders.PyMuPDFLoader`.
- Check `logs.log` in the project root for runtime logs.

■ Contributing & License

Contributions welcome — open an issue or PR with a clear description of the change. This project is licensed under **MIT**.

If you'd like, I can now: - Add a short `CONTRIBUTING.md` and an example GitHub Actions workflow for CI ■ - Add a `Makefile` / `scripts/` helpers to run common flows (scrape → build → query) ■

If any of that sounds useful, tell me which you'd like next!