JVAI Policy Chatbot - Explanation

JVAI Policy Chatbot - Explanation

This project is built for your assignment:

You had to create a chatbot that can read a **policy PDF**, build a searchable database, and answer user questions about the policy with references to the document (page numbers).

It is designed for **beginners**. Below is a file-by-file explanation and then how everything connects.

1. Files in the Project

1) data/policy.pdf
- This is the PDF policy document that the chatbot answers questions about.
- It is parsed (read) page by page.

2) Assignment Task.pdf
- Your assignment brief. (for your reference only, not used in code).

3) requirements.txt
- List of Python libraries you must install (PyMuPDF, FAISS, sentence-transformers, gradio, etc.).

4) utils.py
- Small helper functions.
- `chunk_text()` = splits large text into smaller chunks (so the bot can search better).
- `extract_keywords()` = helps with follow-up questions like "what about debt?".
- `format_sources()` = shows which page numbers the answer came from.

5) ingest.py
- **Step 1: Extract Data from PDF**
- Reads the PDF with PyMuPDF, page by page.
- Splits text into chunks and keeps track of the page number for each chunk.
- **Step 2: Build a Vector Database**
- Uses the embedding model `all-MiniLM-L6-v2` to turn each chunk into a vector (numbers).
- Stores these vectors in FAISS (a special search index for vectors).

- Saves:

- `index/index.faiss` (the actual searchable database)

- `index/meta.json` (the metadata: chunk text + page numbers)

6) chat.py
- The chatbot itself.

- Two modes:

- CLI (command-line): run `python chat.py`
- Gradio web UI: run `python chat.py --ui`

- Workflow:

1. Loads the FAISS index + metadata from `ingest.py`.
2. User asks a question.
3. Question is embedded into a vector.

4. Searches FAISS for the most similar chunks (retrieval).

5. Answers:

- **Offline mode (default):** shows the top excerpts + page numbers directly.
- **Online mode (optional):** if you have an OPENAI_API_KEY, it asks an AI model
to write a concise answer, but only using the retrieved text (with sources).
6. Conversation memory: if your next question is very short ("what about debt?"),
it prepends the last question to give better context.

7) README.md
- A detailed guide (step by step) on how to install, run, and submit the project.

8) EXPLANATION.txt (this file)

- Beginner-friendly explanation of each file and how the whole project works.

2. How the Whole Project Works (Step by Step)

A) Extract Data
- `ingest.py` reads the PDF, cleans the text, splits it into overlapping chunks
(about 700 characters each, with 120 characters overlap), and remembers page numbers.

B) Build Search Database
- Each chunk is converted into a vector using sentence-transformers (MiniLM model).
- FAISS stores all vectors for fast similarity search.

C) Answer Questions
- `chat.py` takes your question → turns it into a vector → searches FAISS for the closest chunks.
- Shows the answer + the page numbers it came from.
- (Optional) If you set an OpenAI API key, an LLM writes a nicer answer but still cites pages.

D) Memory
- Short follow-up questions use the previous question for context.

E) User Interfaces
- CLI: simple Q&A; in terminal.
- Gradio UI: friendly web app at http://127.0.0.1:7860

3. Why This Project? (Purpose)

1. Extract data from a PDF (done in ingest.py with PyMuPDF + chunking).
2. Set up a searchable database (done with FAISS + embeddings).
3. Build a chatbot that can answer based on the document (done in chat.py).

* Learning how to connect PDF data → embeddings → FAISS → chatbot.
* Demonstrating retrieval-based QA (RAG = Retrieval-Augmented Generation).
* Submitting your assignment: everything is included (code, data, instructions).

4. Summary in Simple Words

That's it ■