RAG Chatbot for Customer Support (JioPay)

LLM Production and Deployment — Vizuara

Released: August 28, 2025 **Due:** September 13, 2025 (23:59 IST)

1 Overview

In this assignment, you will build and deploy a production-grade Retrieval-Augmented Generation (RAG) chatbot that automates customer support for **JioPay**. Your chatbot must use only **publicly accessible** JioPay information (business website + help center/FAQs; additional public sources allowed) to answer user queries with high relevance, accuracy, and transparency. You will:

- Construct a knowledge base from JioPay public pages.
- Implement retrieval (vector search) + LLM answer generation with citations.
- Build a clean web UI and deploy it to a public URL.
- Conduct **ablation studies** on chunking, file ingestion, and embeddings.
- Submit a concise PDF report and your code repository.

2 Knowledge Base: Allowed Data Sources

- **JioPay Business Website:** crawl/scrape publicly accessible pages (jiopay.com/business).
- **JioPay Help Center / FAQs:** extract all FAQ content.
- **Any other public sources** that are clearly relevant and citeable.

Compliance: Respect robots.txt and site T&Cs. Do not access gated or user data.

3 System Requirements

3.1 Retrieval-Augmented QA

- **Embeddings + Vector Store:** create dense vectors for text chunks.
- **Retriever:** top-k search with reranking optional.
- **Generator:** an LLM that answers *only* from retrieved context; minimize hallucinations.
- **Citations:** show source URLs/paths for the chunks used.

3.2 Knowledge Base Construction

- **Scrape/Extract:** use a web scraper or manual export to collect content.
- **Normalize:** clean HTML, remove boilerplate, preserve headings/structure.
- **Chunking Strategy:** implement and **compare**:
  1. Fixed chunking (size/overlap grid).
  2. Semantic chunking (sentence/paragraph boundaries via embeddings).
  3. Structural chunking (by headings/HTML tags).
  4. Recursive chunking (hierarchical fallback).
  5. LLM-based chunking (instruction-aware segmentation).

- **Storage:** persist as JSON/CSV and index into a vector DB.

3.3 Frontend Interface

- Chat interface with user prompt, model response, **and citations** (URLs + snippet).
- Show *top-k retrieved chunks* expandable inline.
- Display latency and token usage/cost (if available).

3.4 Hosting and Deployment

- Deploy to a public host (e.g., Vercel, AWS, Azure).

4 Evaluation Criteria (Rubric)

Criterion	**Weight**
Relevance & Accuracy (answers grounded in sources)	30%
Technical Implementation (RAG design, retrieval, indexing)	25%
Ablation Studies (depth, rigor, insights)	20%
Frontend Usability (UX, citations, clarity)	10%
Creativity (chunking/rerank/cloud integrations)	5%
Presentation (report quality, demo clarity)	10%

5 What to Build (Minimum Features)

- **Backend:** ingestion → cleaning → chunking → embedding → vector index → retriever → LLM w/ citations.
- **Frontend:** chat box, streaming answers, expandable citations (URL + title + short snippet), and retrieved-chunk viewer.
- **Deployment:** live URL accessible without auth; environment variables secured; README with run steps.

6 Ablation Design (Required in Report)

6.1 Chunking

Include at minimum:

- **Fixed:** size $\in \{256, 512, 1024\}$ tokens; overlap $\in \{0, 64, 128\}$.
- **Semantic:** sentence/paragraph splits (e.g., similarity-thresholded merges).
- **Structural:** split by headings/HTML tags; preserve hierarchy.
- **Recursive:** fallback from large structural blocks to smaller semantic/fixed chunks.
- **LLM-based:** prompt-guided segmentation; cost vs quality analysis.

6.2 Embeddings

Compare at least three (suggestions):

- *OpenAI* (e.g., `text-embedding-3-*`).
- *E5* (e.g., `intfloat/e5-base`), *BGE/MiniLM*.
- Any strong open model relevant to English support pages.

6.3 Scraping/Ingestion

Compare at least two distinct pipelines. Suggested:

- `requests + BeautifulSoup4` (sitemap/URL list).
- `trafilatura` (readability extraction).
- Headless browser (e.g., Playwright) if dynamic content is present.

Report coverage (#pages/#tokens), cleanliness (noise ratio), throughput (pages/sec), and failure modes (blocked URLs, malformed HTML).

6.4 Reporting Template Tables

Use these tables (expand as needed).

Chunking Ablation (Example Schema):

Strategy	**Size**	**Overlap**	**Top-k P@1**	**Answer F1**	**Latency (ms)**
Fixed	512	64			
Semantic	—	—			
Structural	—	—			
Recursive	—	—			
LLM-based	—	—			

Embeddings Ablation:

Model	**Recall@5**	**MRR**	**Index Size (MB)**	**Avg. Cost / 1k queries**
Model A				
Model B				
Model C				

Ingestion/Scraper Ablation:

Pipeline	**#Pages**	**#Tokens**	**Noise %**	**Throughput**	**Failures (%)**
BS4 (sitemap)					
Trafilatura					
Headless					

7 Evaluation Protocol

- Build a **test set** of $\geq$**10 queries** spanning: onboarding, payments, KYC, refunds, security, pricing, API/integration.
- Report:
  - **Retrieval:** P@1, Recall@k, MRR.
  - **Answer Quality:** exact-match/% supported, LLM-as-judge (faithfulness to citations), and human spot-check notes.
  - **Performance:** latency (P50/P95), cost per query, memory/CPU usage.

8 What to Submit

1) Live Application

- Public URL: `https://<your-app-host>.com`

2) Code Repository

- Repo name: `<team-or-name>-rag-jiopay`
- Must contain: `README.md` (setup, env vars, run/deploy), `LICENSE`, `requirements/environment`, `.env.example`, and a `DATA_CARD.md` (source URLs, counts).

3) PDF Report (Max 8 pages)

Single PDF uploaded to Overleaf-export or the repo `/reports` folder. Use the following structure:

1. **Abstract** (5–7 lines).
2. **System Overview** (diagram + brief description).
3. **Data Collection** (sources, coverage, ethics/compliance).
4. **Chunking Ablation** (design, metrics, results, insights).
5. **Embeddings Ablation** (design, metrics, results, insights).
6. **Ingestion/Scraper Ablation** (design, metrics, results, insights).
7. **Retrieval + Generation** (prompting, top-k, rerankers, guardrails).
8. **Deployment** (infra, costs, monitoring).
9. **Limitations & Future Work**.

Submission Method

Upload to Vizuara Portal.