High Level Architecture

- **System Overview:**
    - The codebase implements a small Retrieval-Augmented Generation (RAG) service: a thin app layer (API) plus a rag library that handles ingestion, document chunking, embeddings, vector storage/retrieval, prompting, and an LLM wrapper. Persistent vector index is stored under job_index.faiss.

- **Components & Responsibilities:**
    - API layer (app) — Files: main.py, schemas.py, __init__.py
        - Provides request/response surface (likely FastAPI or similar).
        - schemas.py defines request/response data models used by the API.
        - main.py wires endpoints to rag retriever/LLM calls and orchestration.
    - RAG library (rag) — Files: chunking.py, embedding.py, llm.py, preprocessing.py, prompt.py, retriever.py, vector_store.py, __init__.py
        - preprocessing.py: text cleaning, normalization, sentence-splitting.
        - chunking.py: converts documents to chunks / passages with overlap and metadata.
        - embedding.py: abstraction to produce vector embeddings (pluggable provider).
        - vector_store.py: FAISS-backed vector index management (persist/load, upsert, search).
        - retriever.py: retrieval strategy (embedding search, optionally hybrid or reranker).
        - prompt.py: prompt templates and assembly for LLM input, including contextual concatenation of retrieved chunks.
        - llm.py: LLM wrapper to call completion/generation APIs (provider-agnostic interface).
    - Scripts & Data:
        - ingest.py — ingestion pipeline to build job_index.faiss from input documents.
        - job_index.faiss — persisted FAISS index file used at runtime.

Engineering Decisions & Reasoning

- Modular separation (app vs rag):
    - Decision: Keep API surface separate from RAG internals.
    - Reasoning: Makes the retrieval/generation pipeline reusable (CLI, worker, tests), and allows swapping or testing the API without changing retrieval logic.
- Single-purpose modules (chunking, embedding, retriever, vector_store, llm, prompt):
    - Decision: Each concern has its own module.
    - Reasoning: Improves testability, replacement (e.g., swap FAISS for Pinecone), and isolation of provider-specific code.
- FAISS persistent index:
    - Decision: Use FAISS file job_index.faiss for vector store.
    - Reasoning: Fast local similarity search, simple persistence as a file for lightweight deployments and fast prototyping without external services.
- Embedding/LLM abstraction layers:

    - Decision: Provide wrapper functions/classes around embedding and LLM providers.
    - Reasoning: Decouples model provider (OpenAI, local LLM, Hugging Face) and eases migration or A/B testing.
- Chunking with overlap & metadata:
    - Decision: Use overlapping chunks and include metadata per chunk.
    - Reasoning: Overlap preserves context across splits and metadata allows filtering and provenance for retrieved passages.
- Prompt templating module:
    - Decision: Keep prompt templates centralized in prompt.py.
    - Reasoning: Easier to tune prompts, apply safety/length heuristics, and support multiple prompt variants per task.
- Simple ingestion script:
    - Decision: Provide ingest.py rather than a complex ETL pipeline.
    - Reasoning: Low friction for getting started; developers can run ingestion locally to build the index.

Setup and Installation Instructions:

Follow the Instructions Provided here: Setup and Installation Instructions

Example Uses: requests and Expected Response:

Request :


```
{
  "query": "What are some software engineering jobs in Los Angeles, CA?",
  "top_k": 2
}
```


Response:


```
{
  "query": "What are some software engineering jobs in Los Angeles, CA?",
  "jobs": [
    {
      "score": 0.4967699348926544,
      "text": "position description job summary saint-gobain research north america is looking for a talented, and energetic data scientist to join our data team in driving digital transformations for key programs. the mission of this team is to develop world-class ai, data science, machine learning, and related solutions to extract insights from data for driving business and customer value. we provide data science expertise and partner with teams across the company, including r&d, sales, marketing, supply chain, and manufacturing. the data scientist will help the organization develop a better understanding of r&d and business strategies by driving solutions and working with the various teams to deliver end-point solutions for improved industrial processes and provide optimized digital experiences. the main responsibilities include: maintains good code development practices and interact with other data teams for code traceability, code practices carries several small projects or
```

studies through to successful completion that involve the extension of present or new technology in one or more areas, to improve and/or development of processes and testing techniques. manages and documents projects effectively providing data and results generates project timelines/plans, technical memos, and research reports, and presents to internal customers, project teams, and/or management required qualifications requirements: bachelor's degree with a minimum of 3+ years' experience or master's degree with thesis or ms with 1+ years of industry experience in computer science, operations research, statistics, or other related disciplines proficient in ms office, and industry-standard statistics and data visualization packages (grafana/powerbi/tableau), machine learning algorithms (supervised, unsupervised, classification, regression), and data structures (sql), and deep learning (pytorch/tensorflow) advanced computer vision techniques (open cv, classification, object detection, segmentation) preferred experience with cloud-based development on azure preferred ci/cd pipelines, mlops experience preferred company summary saint-gobain research (sgr) north america is one of saint-gobain's largest industrial research laboratories. the center is home to more than 400 employees working at the leading edge of innovation on multidisciplinary industrial research, preparing the future by designing tomorrow's products and processes across an amazing variety of markets and technical disciplines. since 1985, sgr north america has been supporting existing businesses, developing future businesses, and nurturing talent. saint-gobain designs, manufactures and distributes materials and solutions which are key ingredients in the wellbeing of each of us and the future of all. they can be found everywhere in our living places and our daily life: in buildings, transportation, infrastructure and in many industrial applications. they provide comfort, performance and safety",

"job_id": "LF0146",
"title": "Data Scientist II",
"company": "Saint-Gobain",
"location": "Westborough, MA",
"level": "Mid Level",
"category": "Data and Analytics",
"tags": null
},
{
"score": 0.4744660258293152,

"text": "working at atlassian atlassians can choose where they work whether in an office, from home, or a combination of the two. that way, atlassians have more control over supporting their family, personal goals, and other priorities. we can hire people in any country where we have a legal entity. interviews and onboarding are conducted virtually, a part of being a distributed-first company. to help our teams work together effectively, this role is fully remote, but requires you to be located in either us pacific or mountain time zones . your future team we're looking for a principal backend software engineer to join our team, passionately focused on delivering creative improvements for our engineering teams. what you'll do regularly unblock challenges faced in the software development cycle, from technical design to launch create solutions that are used by other teams and products - determine plans-of-attack on large projects routinely solve complex architecture challenges and apply architectural standards and start using them on new projects lead code reviews and documentation and take on complex bug fixes, especially on high-risk problems set the standard for meaningful code reviews - partner across engineering teams to take on company-wide projects transfer your depth of knowledge from your current language to

excel as a software engineer mentor more junior members of the team your background 10+ years experience in multiple hands-on software/technology leadership roles, with end-to-end responsibility through the software development lifecycle bachelor's degree with a preference for computer science degree expertise with one or more prominent languages such as java, python, kotlin, go, or scala is required. understanding of saas, paas, iaas industry with hands-on experience with public cloud offerings (e.g., aws, gcp, or azure) knowledge to evaluate trade-offs between correctness, robustness, performance, space and time practice in mentoring other engineers and influencing decision makers throughout the organization consideration of customer impact when making technical decisions if you've got these skills, even better: experience in developer experience or developer productivity - proficiency in java or nodejs experience in at least one additional language experience in ci (continuous integration) and cd (continuous deployment) compensation at atlassian, we strive to design equitable, explainable, and competitive compensation programs. to support this goal, the baseline of our range is higher than that of the typical market range, but in turn we expect to hire most candidates near this baseline. base pay within the range is ultimately determined by a",
    "job_id": "LF0233",
    "title": "Principal Backend Software Engineer",
    "company": "Atlassian",
    "location": "Canada, Flexible / Remote, San Francisco, CA",
    "level": "Senior Level",
    "category": "Software Engineering",
    "tags": null
  }
 ],
 "answer": "Data Scientist II Company: Saint-Gobain Location: Westborough, MA Level: Mid Level Category: Data and Analytics Job Title: Principal Backend Software Engineer Company: Atlassian Location: Canada, Flexible / Remote, San Francisco, CA Level: Senior Level Category: Software Engineering"
}

Assumptions Made During Development

- The API is small-scale and synchronous (typical for initial prototypes); heavy concurrency and streaming are not required.
- job_index.faiss is sufficient as the persistent vector store for intended usage (dataset size fits memory).
- Embeddings and LLM calls are external (network) services or provider SDKs; network reliability, rate limits, and keys are handled outside or via env config.
- Documents to index are text-first (no multimodal support).
- Security, authentication, and rate-limiting are out-of-scope for the initial implementation.
- Config (API keys, model names, vector parameters) is provided via environment variables or simple config constants (not a dedicated config service).
- Minimal telemetry/logging is acceptable for now; deep observability/metrics aren’t yet required.

Drawbacks

- Scalability & durability limits:
    - FAISS file-based index can become memory-constrained for large corpora and lacks built-in multi-node durability.
- Lack of robust configuration & secrets management:
    - If providers and keys are embedded as constants, this is brittle and insecure.
- Synchronous/blocking I/O:
    - If app and rag use blocking calls to external APIs, throughput will be limited under concurrent load.
- Limited error handling & retries:
    - External call failures (embeddings/LLM) may not have retries, backoff, or circuit-breaker behavior.
- No tests or CI hooks included:
    - Harder to maintain correctness and regressions.
- Minimal observability & monitoring:
    - No metrics, tracing or structured logs for latency, error rates, or vector search quality.
- No fine-grained access control or input validation beyond basic schemas:
    - API may be vulnerable to misuse if exposed publicly.
- Potential prompt/context-length issues:
    - If prompt assembly doesn't enforce token limits, calls may fail or cost more.

Future Enhancements

- Vector store improvements:
    - Add pluggable support for managed vector DBs (Pinecone, Milvus, Weaviate) and namespace/collection support.
    - Implement sharding/partitioning and incremental updates instead of single-file rewrite.
- Asynchronous & streaming I/O:
    - Convert app and rag calls to async, enable streaming LLM responses, and add request timeouts.
- Robust config & secrets:
    - Centralize config (12-factor), use dotenv/Vault/KMS for secrets, and validate configs at startup.
- Retry/backoff & resiliency:
    - Add retries, exponential backoff, and circuit breakers for external calls.
- Authentication, authorization, and rate-limiting:
    - Add API auth (JWT, OAuth), role-based access, and per-user rate limits.
- Testing & CI/CD:
    - Add unit tests for chunking, embedding (mocked), retriever, and integration tests for end-to-end flows. Add GitHub Actions / CI.
- Observability:
    - Install structured logging, tracing, and metrics (Prometheus/OTel); add dashboards and alerts.
- Prompt & retrieval improvements:

    - Add reranking models or cross-encoder reranker, support retrieval augmentation (metadata filtering), and implement prompt-length-aware chunk selection.
- Privacy & security:
    - Encrypt persisted indices at rest; anonymize/scrub sensitive PII before storing.
- Batching & cost optimization:
    - Batch embedding/LLM requests to reduce per-call overhead and cost.
- Tooling & developer ergonomics:
    - Provide Dockerfile, docker-compose, and a small README with run/ingest commands and example requests.
- Schema & contract evolution:
    - Add API versioning and extend schemas.py with richer typed models and validation.
- Multimodal & richer metadata support:
    - Support images/audio attachments, vector metadata indexing, and richer provenance linking.