Production-ready RAG helper classes

> **Package root used below:** `com.example.rag`
>
> Change the base package to match your project and add these files to `src/main/java` under the corresponding sub-packages.

---

1. `TieredApiVectorStoreCache.java`

```java
package com.example.rag.cache;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.github.benmanes.caffeine.cache.Cache;
import com.github.benmanes.caffeine.cache.Caffeine;
import com.github.benmanes.caffeine.cache.LoadingCache;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.embedding.LocalHuggingFaceEmbeddingModel;
import dev.langchain4j.store.embedding.EmbeddingStore;
import dev.langchain4j.store.embedding.EmbeddingStoreIngestor;
import dev.langchain4j.store.embedding.inmemory.InMemoryEmbeddingStore;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.web.client.RestClientException;
import org.springframework.web.client.RestTemplate;

import java.time.Duration;
import java.util.List;
import java.util.stream.StreamSupport;

/**
 * Two-tier cache (short-lived burst level + long-lived level) that builds and
 * returns an {@link EmbeddingStore} for any REST endpoint.
 */
public class TieredApiVectorStoreCache {

    private static final Logger log =
LoggerFactory.getLogger(TieredApiVectorStoreCache.class);

    private final RestTemplate rest;
    private final ObjectMapper mapper;
    private final EmbeddingModel embed;
    private final Cache<String, EmbeddingStore<Document>> shortCache;
    private final LoadingCache<String, EmbeddingStore<Document>> longCache;

    public TieredApiVectorStoreCache(RestTemplate restTemplate,
                                     ObjectMapper mapper,
                                     long shortTtlSec,
                                     long shortMaxEntries,
                                     long longTtlMin,
                                     long longMaxEntries) {
        this.rest = restTemplate;
        this.mapper = mapper;
        this.embed = LocalHuggingFaceEmbeddingModel.builder()
                .modelName("sentence-transformers/all-MiniLM-L6-v2")
                .build();

        this.shortCache = Caffeine.newBuilder()
                .expireAfterWrite(Duration.ofSeconds(shortTtlSec))
                .maximumSize(shortMaxEntries)
                .build();

        this.longCache = Caffeine.newBuilder()
                .expireAfterWrite(Duration.ofMinutes(longTtlMin))
                .maximumSize(longMaxEntries)
                .build(this::buildStore);
    }

    /**
     * Return a ready-to-search {@link EmbeddingStore}. Always non-null.
     */
    public EmbeddingStore<Document> get(String url) {
        EmbeddingStore<Document> store = shortCache.getIfPresent(url);
        if (store == null) {
            store = longCache.get(url);          // may trigger loader
            shortCache.put(url, store);          // seed burst layer
        }
        return store;
    }

    /** Trigger an asynchronous rebuild for the given URL (used by scheduler).
*/
    public void refresh(String url) {
        longCache.refresh(url);
        shortCache.invalidate(url);
        log.debug("Cache refresh triggered for {}", url);
    }
```

```java
    /* -------------------- internal helpers -------------------- */

    private EmbeddingStore<Document> buildStore(String url) {
        try {
            String json = rest.getForObject(url, String.class);
            List<Document> docs = toDocuments(url, json);

            EmbeddingStore<Document> store = new InMemoryEmbeddingStore<>();
            EmbeddingStoreIngestor.builder()
                    .embeddingModel(embed)
                    .embeddingStore(store)
                    .build()
                    .ingest(docs);

            log.info("Built vector store for {} ({} docs, {} B)", url,
docs.size(),
                     json == null ? 0 : json.length());
            return store;
        } catch (RestClientException e) {
            log.warn("Failed to fetch {} – returning empty store", url, e);
            return new InMemoryEmbeddingStore<>();
        } catch (Exception ex) {
            throw new IllegalStateException("Cannot build vector store for " +
url, ex);
        }
    }

    private List<Document> toDocuments(String url, String json) throws
Exception {
        JsonNode root = mapper.readTree(json);
        JsonNode data = root.path("data").isMissingNode() ? root :
root.path("data");
        return StreamSupport.stream(data.spliterator(), false)
                .map(node -> Document.builder()
                        .text(node.isContainerNode() ? node.toPrettyString() :
node.asText())
                        .metadata("source", url)
                        .build())
                .toList();
    }
}
```

2. `StaticEndpointReloader.java`

```java
package com.example.rag.cache;

import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.util.Map;

/**
 * Warms up & periodically refreshes static endpoints defined in YAML.
 */
@Slf4j
@Component
@RequiredArgsConstructor
public class StaticEndpointReloader {

    private final TieredApiVectorStoreCache cache;

    @Value("#{${rag.rest-api.static:{}}}")
    private Map<String, String> staticEndpoints;

    @PostConstruct
    public void warmUp() {
        staticEndpoints.values().forEach(url -> {
            cache.get(url);      // build & cache once
            log.info("Pre-loaded static endpoint {}", url);
        });
    }

    @Scheduled(cron = "${rag.rest-api.static-refresh-cron:0 0 * * * *}")
    public void refresh() {
        staticEndpoints.values().forEach(cache::refresh);
        log.debug("Static endpoints refreshed");
    }
}
```

Requires `spring-boot-starter` + `spring-boot-starter-aop` + `spring-boot-starter-scheduler` *(enabled by* `@EnableScheduling` *in your application class).*

2b. `StaticDocumentReloader.java`

```java
package com.example.rag.cache;

import jakarta.annotation.PostConstruct;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

import java.util.Map;

/**
 * Pre-loads static markdown / PDF paths at application startup and refreshes
 * them on a configurable cron schedule.
 */
@Slf4j
@Component
@RequiredArgsConstructor
public class StaticDocumentReloader {

    private final DocumentVectorStoreCache docCache;

    @Value("#{${rag.doc.static:{}}}")
    private Map<String, String> staticDocs;      // name → path

    @PostConstruct
    public void warmUp() {
        staticDocs.values().forEach(p -> {
            docCache.get(p);
            log.info("Pre-loaded static doc {}", p);
        });
    }

    @Scheduled(cron = "${rag.doc.static-refresh-cron:0 0 * * * *}")
    public void refresh() {
        staticDocs.values().forEach(docCache::refresh);
        log.debug("Static documents refreshed");
    }
}
```

Like the endpoint reloader, this bean requires `@EnableScheduling` *in your main application class.*

3. `MultiApiSimilarityRetriever.java`

```java
package com.example.rag.retriever;

import com.example.rag.cache.TieredApiVectorStoreCache;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.rag.content.retriever.ContentRetriever;
import dev.langchain4j.rag.content.retriever.EmbeddingStoreContentRetriever;
import dev.langchain4j.rag.query.Query;
import dev.langchain4j.store.embedding.EmbeddingStore;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.util.*;
import java.util.stream.Collectors;

/**
 * Aggregates similarity search across configurable static endpoints and
 * per-request dynamic URLs supplied via Query metadata.
 */
@Component
public class MultiApiSimilarityRetriever implements ContentRetriever {

    private static final Logger log =
LoggerFactory.getLogger(MultiApiSimilarityRetriever.class);

    private final Map<String, String> staticEndpoints; // name → URL
    private final TieredApiVectorStoreCache cache;
    private final EmbeddingModel embed;
    private final int topK;

    public MultiApiSimilarityRetriever(@Value("#{${rag.rest-api.static:{}}}")
Map<String, String> staticEndpoints,
                                        TieredApiVectorStoreCache cache,
                                        EmbeddingModel embed,
                                        @Value("${rag.rest-api.top-k:6}") int
topK) {
        this.staticEndpoints = staticEndpoints;
        this.cache = cache;
        this.embed = embed;
        this.topK = topK;
    }
```

```java
@Override
public List<Document> retrieve(Query query) {
    Set<String> targetUrls = new HashSet<>();

    // 1️⃣ static endpoint selection
    endpointNamesFor(query).stream()
            .map(staticEndpoints::get)
            .filter(Objects::nonNull)
            .forEach(targetUrls::add);

    // 2️⃣ raw URLs via metadata("urls", "http://…")
    Object dyn = query.metadata("urls");
    if (dyn instanceof String s) {
        Arrays.stream(s.split(","))
                .map(String::trim)
                .filter(str -> str.startsWith("http"))
                .forEach(targetUrls::add);
    }

    if (targetUrls.isEmpty()) {
        log.debug("No endpoints selected – returning empty context");
        return List.of();
    }

    // 3️⃣ parallel similarity search per URL
    List<Document> combined = targetUrls.parallelStream()
            .flatMap(url -> {
                EmbeddingStore<Document> store = cache.get(url);
                return EmbeddingStoreContentRetriever.builder()
                        .embeddingStore(store)
                        .embeddingModel(embed)
                        .maxResults(topK * 2)
                        .build()
                        .retrieve(query)
                        .stream();
            })
            .distinct()      // de-dupe identical text
            .limit(topK)     // global top-k
            .toList();

    return combined;
}

private Set<String> endpointNamesFor(Query q) {
    Object meta = q.metadata("endpoints");
    if (meta instanceof String s) {
        return Arrays.stream(s.split(","))
```

```
                    .map(String::trim)
                    .filter(staticEndpoints::containsKey)
                    .collect(Collectors.toSet());
        }
        return staticEndpoints.keySet();
    }
}
```

---

4. `RagConfig.java`

```java
package com.example.rag.config;

import com.example.rag.cache.TieredApiVectorStoreCache;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.embedding.LocalHuggingFaceEmbeddingModel;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;
import com.fasterxml.jackson.databind.ObjectMapper;

@Configuration
public class RagConfig {

    /* Shared lightweight embedding model – loaded once per JVM */
    @Bean
    public EmbeddingModel localMiniLm() {
        return LocalHuggingFaceEmbeddingModel.builder()
                .modelName("sentence-transformers/all-MiniLM-L6-v2")
                .build();
    }

    @Bean
    public TieredApiVectorStoreCache vectorStoreCache(RestTemplate rest,
                                                      ObjectMapper mapper,
                                                      @Value("${rag.rest-
api.short-ttl-seconds:90}") long shortTtl,
                                                      @Value("${rag.rest-
api.short-max-entries:200}") long shortMax,
                                                      @Value("${rag.rest-
api.long-ttl-minutes:60}") long longTtl,
                                                      @Value("${rag.rest-
api.long-max-entries:100}") long longMax) {
        return new TieredApiVectorStoreCache(rest, mapper, shortTtl, shortMax,
longTtl, longMax);
    }
}
```

---

5. Example `application.yml` snippet

```
spring:
  main:
    allow-bean-definition-overriding: true

rag:
  rest-api:
    static:
      users:      https://reqres.in/api/users?page=2
      settings:   https://api.example.com/v1/settings
    top-k: 6
    parallelism: 4
    short-ttl-seconds: 90
    short-max-entries: 200
    long-ttl-minutes: 60
    long-max-entries: 100
    static-refresh-cron: "0 */30 * * * *"   # every 30 minutes
```

How to integrate

1. **Enable scheduling** in your Spring Boot application:

```
@SpringBootApplication
@EnableScheduling
public class RagApplication { public static void main(String[] args) {
    SpringApplication.run(RagApplication.class, args);
}}
```

2. **Add** the `MultiApiSimilarityRetriever` instance to your LangChain4j `Assistant`.
3. **Pass dynamic URLs** via `Query` metadata when needed:

```
Query q = Query.from(question)
               .withMetadata("urls", "https://api.example.com/user/42/
bookings");
```

You now have production-ready, pluggable RAG components that:

- pre-embed static endpoints at startup,
- refresh them on a cron,
- embed dynamic endpoints on demand,
- cache everything through a high-performance two-tier Caffeine strategy.

---

6. `DocumentVectorStoreCache.java`

```java
package com.example.rag.cache;

import com.github.benmanes.caffeine.cache.*;
import com.fasterxml.jackson.databind.ObjectMapper;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.data.document.loader.FileSystemDocumentLoader;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.model.embedding.LocalHuggingFaceEmbeddingModel;
import dev.langchain4j.store.embedding.*;
import dev.langchain4j.store.embedding.inmemory.InMemoryEmbeddingStore;
import lombok.extern.slf4j.Slf4j;
import org.springframework.util.StringUtils;

import java.nio.file.Path;
import java.time.Duration;
import java.util.List;

/**
 * Two-tier Caffeine cache for *file-based* documents (Markdown, PDF, etc.).
 * Key is an absolute directory/file path.
 */
@Slf4j
public class DocumentVectorStoreCache {

    private final EmbeddingModel embed;
    private final Cache<String, EmbeddingStore<Document>> shortCache;
    private final LoadingCache<String, EmbeddingStore<Document>> longCache;

    public DocumentVectorStoreCache(long shortTtlSec, long shortMax,
                                    long longTtlMin, long longMax) {
        this.embed = LocalHuggingFaceEmbeddingModel.builder()
                .modelName("sentence-transformers/all-MiniLM-L6-v2")
                .build();

        this.shortCache = Caffeine.newBuilder()
                .expireAfterWrite(Duration.ofSeconds(shortTtlSec))
                .maximumSize(shortMax)
                .build();

        this.longCache = Caffeine.newBuilder()
                .expireAfterWrite(Duration.ofMinutes(longTtlMin))
                .maximumSize(longMax)
                .build(this::buildStore);
    }

    public EmbeddingStore<Document> get(String path) {
        EmbeddingStore<Document> store = shortCache.getIfPresent(path);
        if (store == null) {
            store = longCache.get(path);
            shortCache.put(path, store);
        }
        return store;
    }

    public void refresh(String path) {
        longCache.refresh(path);
        shortCache.invalidate(path);
    }

    /* ---------------- helpers ---------------- */
    private EmbeddingStore<Document> buildStore(String pathStr) {
        try {
            Path path = Path.of(pathStr);
            List<Document> docs = FileSystemDocumentLoader.loadDocuments(path);

            EmbeddingStore<Document> store = new InMemoryEmbeddingStore<>();
            EmbeddingStoreIngestor.builder()
                    .embeddingModel(embed)
                    .embeddingStore(store)
                    .build()
                    .ingest(docs);
            log.info("Built doc vector store for {} ({} docs)", path,
docs.size());
            return store;
        } catch (Exception e) {
            log.error("Failed to load documents from {}", pathStr, e);
            return new InMemoryEmbeddingStore<>();
        }
    }
}
```

7. `MultiDocSimilarityRetriever.java`

```
package com.example.rag.retriever;

import com.example.rag.cache.DocumentVectorStoreCache;
import dev.langchain4j.data.document.Document;
import dev.langchain4j.model.embedding.EmbeddingModel;
import dev.langchain4j.rag.content.retriever.ContentRetriever;
import dev.langchain4j.rag.content.retriever.EmbeddingStoreContentRetriever;
import dev.langchain4j.rag.query.Query;
import dev.langchain4j.store.embedding.EmbeddingStore;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.util.*;
import java.util.stream.Collectors;

/**
 * Similar to {@code MultiApiSimilarityRetriever} but for unstructured files.
 */
@Component
public class MultiDocSimilarityRetriever implements ContentRetriever {

    private final Map<String, String> staticPaths;          // name → dir/file
path
    private final DocumentVectorStoreCache cache;
    private final EmbeddingModel embed;
    private final int topK;

    public MultiDocSimilarityRetriever(@Value("#{${rag.doc.static:{}}}")
Map<String,String> staticPaths,
                                        DocumentVectorStoreCache cache,
                                        EmbeddingModel embed,
                                        @Value("${rag.rest-api.top-k:6}") int
topK) {
        this.staticPaths = staticPaths;
        this.cache = cache;
        this.embed = embed;
        this.topK = topK;
    }

    @Override
    public List<Document> retrieve(Query query) {
        Set<String> targetPaths = new HashSet<>();

        // static names from metadata("docs", "handbook,policies")
        Object meta = query.metadata("docs");
        if (meta instanceof String s) {
            Arrays.stream(s.split(","))
                    .map(String::trim)
                    .filter(staticPaths::containsKey)
                    .map(staticPaths::get)
                    .forEach(targetPaths::add);
        } else {
            targetPaths.addAll(staticPaths.values());
        }

        // raw dynamic path via metadata("path", "./uploads/123.md")
        Object dyn = query.metadata("path");
        if (dyn instanceof String s && !s.isBlank()) {
            targetPaths.add(s.trim());
        }

        return targetPaths.parallelStream()
                .flatMap(p -> {
                    EmbeddingStore<Document> store = cache.get(p);
                    return EmbeddingStoreContentRetriever.builder()
                            .embeddingStore(store)
                            .embeddingModel(embed)
                            .maxResults(topK * 2)
                            .build()
                            .retrieve(query)
                            .stream();
                })
                .distinct()
                .limit(topK)
                .toList();
    }
}
```

---

8. `CombinedContentRetriever.java`

```
package com.example.rag.retriever;

import dev.langchain4j.data.document.Document;
import dev.langchain4j.rag.content.retriever.ContentRetriever;
import dev.langchain4j.rag.query.Query;

import java.util.*;
import java.util.stream.Collectors;
```

```java
/**
 * Wraps multiple {@link ContentRetriever}s and merges their results.
 */
public class CombinedContentRetriever implements ContentRetriever {

    private final List<ContentRetriever> delegates;
    private final int topK;

    public CombinedContentRetriever(int topK, ContentRetriever... retrievers) {
        this.delegates = List.of(retrievers);
        this.topK = topK;
    }

    @Override
    public List<Document> retrieve(Query query) {
        return delegates.parallelStream()
                .flatMap(r -> r.retrieve(query).stream())
                .collect(Collectors.toMap(Document::text, d -> d, (a,b) -> a))
                .values()
                .stream()
                .limit(topK)
                .toList();
    }
}
```

---

9. Extended `RagConfig.java` additions

```java
@Bean
public DocumentVectorStoreCache docVectorStoreCache(
        @Value("${rag.rest-api.short-ttl-seconds:90}") long sTtl,
        @Value("${rag.rest-api.short-max-entries:200}") long sMax,
        @Value("${rag.rest-api.long-ttl-minutes:60}") long lTtl,
        @Value("${rag.rest-api.long-max-entries:100}") long lMax) {
    return new DocumentVectorStoreCache(sTtl, sMax, lTtl, lMax);
}

@Bean
public CombinedContentRetriever combinedRetriever(MultiApiSimilarityRetriever
api,
                                                  MultiDocSimilarityRetriever
docs,
                                                  @Value("${rag.rest-api.top-k:
6}") int topK) {
    return new CombinedContentRetriever(topK, api, docs);
}
```

---

10. YAML additions for documents

```
rag:
  doc:
    static:
      handbook:  classpath:docs/employee-handbook.md
      policies:  ./legal/policies/
    static-refresh-cron: "0 */60 * * * *"   # hourly reload of markdown/pdfs
```

Now you have:

- `` — burst + long cache for any file path
- `` — similarity search over markdown/PDF paths
- `` — merges API + document hits and returns global top-k

Wire the new `combinedRetriever` into your `Assistant` instead of the individual ones, and you're ready to RAG across *both* REST data **and** unstructured files with one call.