**Redis Cache Documentation for Exact Cache & Semantic Cache**

**Overview**

This document provides a comprehensive guide on the implementation and usage of Redis for caching within your application. It covers the details of two primary caching mechanisms: **Exact Cache** and **Semantic Cache**. The document also discusses best practices for dynamic TTL management, manual cache invalidation, and monitoring using a metrics collection system.

**1. Exact Cache**

**1.1. Purpose**

The Exact Cache stores responses for queries that are repeated verbatim. It provides quick retrieval of data for identical queries, reducing the need to reprocess or fetch the same data repeatedly from the backend.

**1.2. Implementation Details**

**Pseudocode Outline:**

pseudocode

Copy code

CLASS ExactCacheService:

VARIABLES:

redis\_client // Redis client instance

cache\_hits // Counter for cache hits

cache\_misses // Counter for cache misses

query\_hit\_counts // Dictionary to track hit counts for queries

FUNCTION cacheExactResponse(query, response):

SET cache\_key TO hash(query)

SET ttl TO determineTTL(query)

STORE response IN redis\_client WITH cache\_key AND ttl

LOG "Cached exact response for query"

FUNCTION getExactCachedResponse(query):

SET cache\_key TO hash(query)

GET cached\_response FROM redis\_client WITH cache\_key

IF cached\_response IS NOT null:

INCREMENT cache\_hits

INCREMENT query\_hit\_counts[query]

LOG "Exact cache hit for query"

RETURN cached\_response

ELSE:

INCREMENT cache\_misses

LOG "Exact cache miss for query"

RETURN null

FUNCTION invalidateCacheForQuery(query):

SET cache\_key TO hash(query)

DELETE cache\_key FROM redis\_client

LOG "Invalidated cache for query"

FUNCTION determineTTL(query):

SET hits TO query\_hit\_counts[query] OR 0

IF hits > 10:

RETURN 2 hours

ELSE IF query LENGTH < 50:

RETURN 30 minutes

ELSE:

RETURN 1 hour

**2. Semantic Cache**

**2.1. Purpose**

The Semantic Cache is designed to store responses for queries that are semantically similar. It allows for retrieval of cached responses even when the query is rephrased or slightly altered by leveraging embeddings and semantic similarity.

**2.2. Implementation Details**

**Pseudocode Outline:**

pseudocode

Copy code

CLASS SemanticCacheService:

VARIABLES:

redis\_client // Redis client instance

embedding\_service // Service to generate embeddings

cache\_hits // Counter for cache hits

cache\_misses // Counter for cache misses

FUNCTION cacheSemanticResponse(query, response):

SET embedding TO embedding\_service.generateEmbedding(query)

SET hash\_of\_embedding TO hash(embedding)

SET ttl TO determineTTL(query)

STORE embedding AND response IN redis\_client WITH hash\_of\_embedding AND ttl

LOG "Cached semantic response for query"

FUNCTION getSemanticCachedResponse(query):

SET query\_embedding TO embedding\_service.generateEmbedding(query)

SET hash\_of\_embedding TO hash(query\_embedding)

GET cached\_response FROM redis\_client WITH hash\_of\_embedding

IF cached\_response IS NOT null:

INCREMENT cache\_hits

LOG "Semantic cache hit for query"

RETURN cached\_response

ELSE:

INCREMENT cache\_misses

LOG "Semantic cache miss for query"

RETURN null

FUNCTION invalidateCacheForQuery(query):

SET embedding TO embedding\_service.generateEmbedding(query)

SET semantic\_cache\_key TO hash(embedding)

DELETE semantic\_cache\_key FROM redis\_client

LOG "Invalidated semantic cache for query"

FUNCTION determineTTL(query):

IF query LENGTH < 50:

RETURN 30 minutes

ELSE:

RETURN 1 hour

**3. Redis Configuration**

To ensure optimal performance, Redis should be configured with appropriate eviction policies and memory management strategies.

**Eviction Policy:**

* **LRU (Least Recently Used)**: This policy evicts the least recently used keys when the maximum memory limit is reached.

**Pseudocode Configuration:**

pseudocode

Copy code

SET redis\_max\_memory TO "2gb"

SET redis\_eviction\_policy TO "allkeys-lru"

**4. Monitoring and Metrics**

Integrating monitoring allows for tracking the performance and efficiency of the cache. Metrics like cache hits, misses, and eviction rates should be monitored.

**Monitoring Tool:** Use a metrics collection system with dashboards for visualization.

**Key Metrics:**

* cache.hits.exact: Tracks the number of cache hits for exact cache.
* cache.misses.exact: Tracks the number of cache misses for exact cache.
* cache.hits.semantic: Tracks the number of cache hits for semantic cache.
* cache.misses.semantic: Tracks the number of cache misses for semantic cache.

**Pseudocode for Metrics Collection:**

pseudocode

Copy code

CLASS ExactCacheService:

FUNCTION getExactCachedResponse(query):

SET cache\_key TO hash(query)

GET cached\_response FROM redis\_client WITH cache\_key

IF cached\_response IS NOT null:

INCREMENT cache\_hits

RETURN cached\_response

ELSE:

INCREMENT cache\_misses

RETURN null

CLASS SemanticCacheService:

FUNCTION getSemanticCachedResponse(query):

SET query\_embedding TO embedding\_service.generateEmbedding(query)

SET hash\_of\_embedding TO hash(query\_embedding)

GET cached\_response FROM redis\_client WITH hash\_of\_embedding

IF cached\_response IS NOT null:

INCREMENT cache\_hits

RETURN cached\_response

ELSE:

INCREMENT cache\_misses

RETURN null

**5. Testing and Validation**

Testing ensures the reliability of your caching implementation.

**Unit Tests:**

* Test individual methods in the cache services to validate functionality, TTL handling, and cache invalidation.

**Integration Tests:**

* Verify that the caching layers integrate smoothly with the rest of your application, providing the expected performance improvements.

**Performance Testing:**

* Conduct load testing to assess how the cache performs under various conditions, ensuring that Redis handles the expected volume of traffic efficiently.

**Pseudocode for Testing:**

pseudocode

Copy code

TEST ExactCacheService:

FUNCTION testCacheExactResponse():

CALL cacheExactResponse(query, response)

VERIFY response IS cached WITH correct TTL

FUNCTION testGetExactCachedResponse():

SET cache\_key TO hash(query)

STORE response IN redis\_client WITH cache\_key

GET cached\_response FROM getExactCachedResponse(query)

VERIFY cached\_response EQUALS response

FUNCTION testInvalidateCacheForQuery():

CALL invalidateCacheForQuery(query)

VERIFY cache\_key IS deleted FROM redis\_client

TEST SemanticCacheService:

FUNCTION testCacheSemanticResponse():

CALL cacheSemanticResponse(query, response)

VERIFY response IS cached WITH correct TTL

FUNCTION testGetSemanticCachedResponse():

SET embedding TO embedding\_service.generateEmbedding(query)

SET hash\_of\_embedding TO hash(embedding)

STORE response IN redis\_client WITH hash\_of\_embedding

GET cached\_response FROM getSemanticCachedResponse(query)

VERIFY cached\_response EQUALS response

FUNCTION testInvalidateCacheForQuery():

CALL invalidateCacheForQuery(query)

VERIFY hash\_of\_embedding IS deleted FROM redis\_client

**Conclusion**

By implementing these Redis caching strategies with Exact and Semantic caches, your application will benefit from improved response times and reduced backend load. The integration of best practices such as dynamic TTL management, manual cache invalidation, and robust monitoring will ensure that your caching solution is both efficient and reliable.

This document should serve as a guide for developers working with Redis cache in your application. Ensure continuous monitoring and optimization of the caching strategy based on the application’s evolving needs.