In-Memory Cache with Rest Interface

POWERED BY N|Solid

build passing

The objective of this project is to create an in-memory cache with a Rest interface. The cache is responsible to store JSON objects (strings) in the server’s memory that can be accessed via the API. For simplicity, we assume that the server is single-threaded. The cache should accept the following configuration parameters:

- Number of slots (int, default 10,000): Maximum number of objects to be stored in the server’s memory.
  - Time-To-Live (int, default: 3600 secs): Object’s default time-to-live value in seconds if no TTL is specified as part of a write request.
  - Eviction Policy (enum, default: REJECT): This indicates what to do when the cache runs out of slots. The following options are supported:
  - - OLDEST_FIRST: If there are no slots available the cache will evict the oldest active object and store the new object in its place
  - - NEWEST_FIRST: If there are no slots available the cache will evict the newest active object first and store the new object in its place
  - - REJECT: When the cache runs out of storage it just reject the store request

Flask server provides an HTTP API service that supports the following operations:

- GET /object/{key}
- - This will return the object stored at {key} if the object is not expired.
- - Returns:
    - 200: If the object is found and not-expired
    - 404: If the object is not found or expired
- POST or PUT /object/{key}?ttl={ttl}
- - This will insert the {object} provided in the body of the request into a slot in memory at {key}.
- - Returns:
    - 200: If the server was able to store the object
    - 507: If the server has no storage
- DELETE /object/{key}
- - This will delete the object stored at slot {key}
- - Returns:
    - 200: If the object at {key} was found and removed

        - 404: If the object at {key} was not found or expired

Features

- LRU cache concept was used to store JSON objects in the cache memory.
- Binary Search Tree concept was used to provide quick sort/get/delete access based on TTL.
- Three Eviction policy has been provided depending on the user configuration.

Future imporvement

- Provide unit tests as part of the development process.
- Provide more efficient functionality to take into account TTL.