

CACHE LAB



cachelab ?

cache lab est une (custom in memory key value cache) qui marche avec REST API comme redis ! ca stock les data dans la RAM pour un access rapid !

Pour faire cette base de données en mémoire, j'ai utilisé une hashmap avec des opérations en $O(1)$, ce qui permet d'accéder aux données très rapidement.

hashmap :

hashmap (hash table) c'est la structure des data qui stock key-value pairs et permettre de fair (insert - find - update- delete) rapidement !! grace a $O(1)$

$O(1)$

c'est une operation qui toujours prendre le même temp pour trouver la valeur a partir d'une clé, peu importe la taille des données !

architecture de la hashmap

hash-function

stock un clé (string) dans un bucket (number)

comment ca marche ?

On commence avec hash = 0

On parcourt chaque caractère de la clé

Pour chaque caractère :

1.

- On récupère son code : `charCodeAt(i)`
- On met à jour le hash : $\text{hash} = ((\text{hash} \ll 5) - \text{hash}) + \text{char}$. (djb2-style hash)
- On convertit en entier 32 bits : $\text{hash} = \text{hash} \& \text{hash}$

2.

3. On retourne : `Math.abs(hash) % bucketCount`

▼ core

TS BucketManager.ts

TS DiskPersistence.ts

TS HashFunction.ts

TS HashMapStore.ts

TS StringUtils.ts

bucketManager

Gère le tableau des buckets en mémoire (RAM) chaque bucket est un tableau pouvant contenir plusieurs paires clé-valeur (chainning)

comment ca marche ?

Des méthodes comme set, get et delete permettent d'ajouter, récupérer et supprimer des paires clé-valeur dans les buckets

exemple :

1. `setInBucket(bucketIndex, key, value)` : cherche un key , si il n ya pas elle le fait
2. `getFromBucket(bucketIndex, key)` : recherche un key et si elle le trouve elle return value ou undefined

HashMapStore : le coordinateur principal

Ce composant central orchestre l'ensemble des modules de la hashmap.
Il expose l'API publique et assure la coordination des opérations set, get et delete.
Il fait le lien entre la logique métier et le stockage en mémoire

HashMapStore : le coordinateur principal

Ce composant central orchestre l'ensemble des modules de la hashmap.
Il expose l'API publique et assure la coordination des opérations set, get et delete.
Il fait le lien entre la logique métier et le stockage en mémoire

key features pour le hashmap

O(1) Performance

- Calcul du bucket via la fonction de hachage → accès direct `buckets[index]`
- Opérations get, set, delete en O(1) en moyenne
- Même temps d'accès pour 10 ou 10M de clés

2. Gestion des Collisions – Chaînage

- Collision = plusieurs clés dans le même bucket
- Chaque bucket contient un tableau de paires clé-valeur
- Recherche locale dans le bucket (1–2 éléments en moyenne)
- Performance toujours O(1) en pratique

key features pour le hashmap

Load Factor & Redimensionnement

- Load factor = nombre d'éléments / nombre de buckets
- Seuil fixé à 0.75
- Dépassement → doublement automatique des buckets + rehash
- Garantit la performance $O(1)$ à long terme