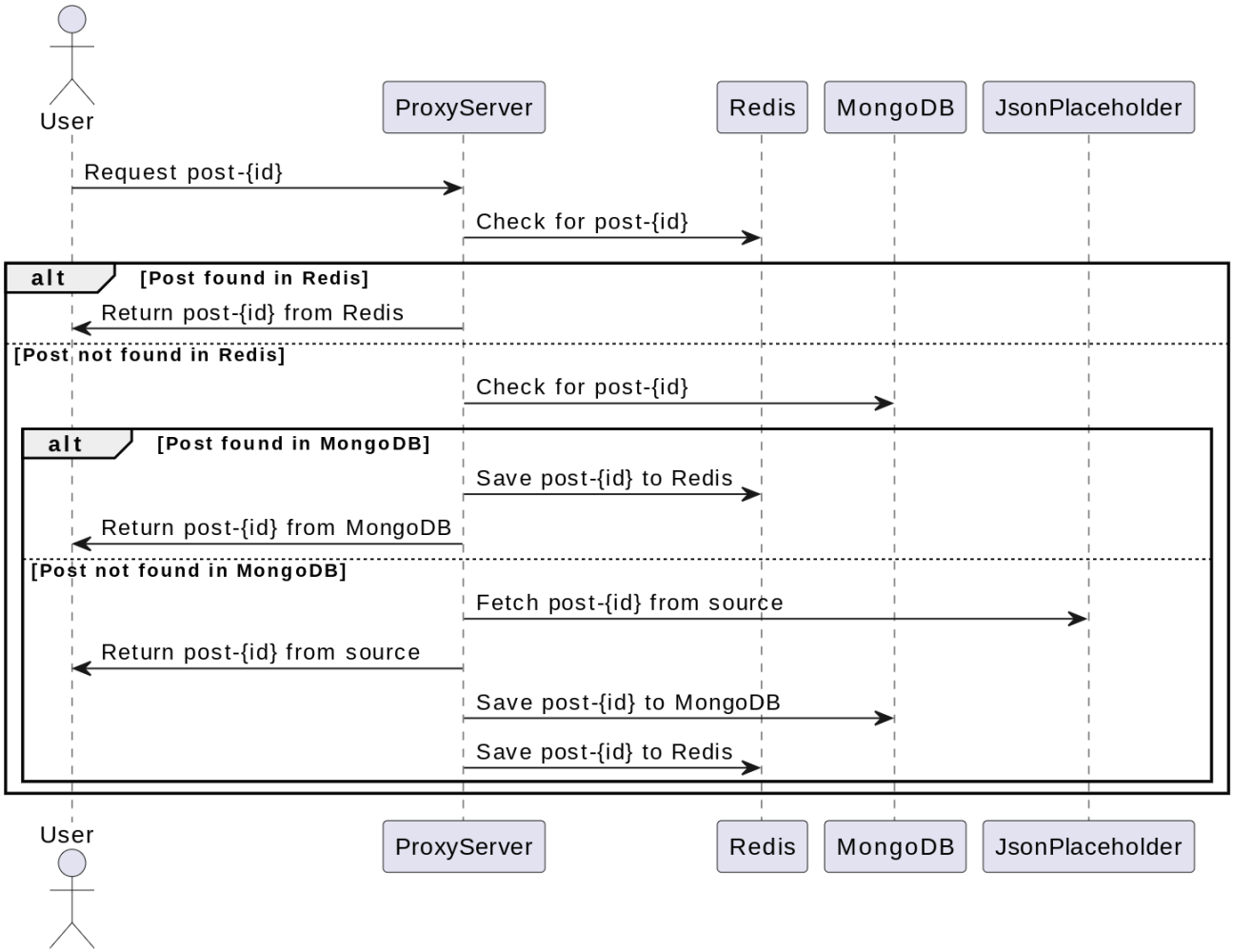
Rapport de projet Redis MongoDB

Introduction

Le code fourni est un programme Go qui expose un point de terminaison HTTP à `/getPost` pour récupérer et mettre en cache des données JSON depuis l'API JSONPlaceholder. L'application utilise une cache Redis pour améliorer l'efficacité des opérations, tandis que les données sont également persistées dans une base de données MongoDB. Ce rapport examine la structure du code, mettant en évidence les fonctionnalités principales, y compris l'intégration avec MongoDB, et propose des améliorations potentielles.



Pourquoi Go?

La décision d'utiliser le langage de programmation Go pour ce projet repose sur deux principaux avantages : la gestion robuste des erreurs et la prise en charge native de la concurrence.

1. Gestion des Erreurs :Cela permet une gestion claire et explicite des erreurs à chaque étape du processus. Les erreurs sont traitées de manière précise, offrant

une visibilité complète sur les éventuels problèmes.

2. **Concurrence** :
   - **Goroutines :** Go offre un support natif pour les goroutines, permettant une exécution concurrente légère. Dans les fonctions `saveToMongo` et `saveToRedis`, l'utilisation de goroutines est illustrée avec les déclarations `go saveToMongo(id, result)` et `go saveToRedis(redisKey+id, result)`.
   - **Avantage :** La concurrence facilite le traitement asynchrone des opérations, améliorant ainsi les performances en permettant à certaines tâches de s'exécuter en parallèle. Cela est particulièrement avantageux dans le contexte des opérations de base de données et des requêtes réseau, réduisant les temps d'attente.

Structure du Code

1. **Fonctionnalité Principale** : La fonction principale initialise un serveur HTTP pour gérer les requêtes sur le port 3333, permettant au serveur de répondre aux requêtes entrantes et de gérer le point de terminaison `/getPost`.
2. **Gestionnaire `getPost`** : Cette fonction traite les requêtes à `/getPost`, récupérant les données depuis la cache Redis ou la base de données MongoDB, en fonction de la disponibilité. Elle mesure le temps d'exécution et renvoie la réponse JSON mise en forme. Le code doit mettre davantage en évidence les avantages de l'utilisation combinée de la cache et de la base de données MongoDB pour améliorer les performances.
3. **Opérations Redis** :
   - `saveToRedis` : Enregistre une paire clé-valeur (ID de post et données JSON) dans Redis avec une expiration de 5 secondes. Cela permet d'éviter des requêtes fréquentes à l'API externe.
   - `getFromRedis` : Récupère les données JSON depuis Redis en fonction de l'ID du post, optimisant ainsi les performances en cas de cache hit.
4. **Opérations MongoDB** :
   - `saveToMongo` : Stocke les données JSON dans MongoDB, assurant la persistance des données pour les futures requêtes.
   - `getFromMongo` : Récupère les données depuis MongoDB en fonction de l'ID du post. Si les données ne sont pas présentes dans le cache Redis, cette opération est cruciale.
5. **Fonctions de Formatage JSON** :
   - `prettifyJSON` : Met en forme les données JSON pour une meilleure lisibilité, bien que cette fonction ne soit pas explicitement définie dans le code.
   - `mapToString` : Convertit une carte en une chaîne formatée en JSON, utilisée pour la réponse HTTP.
6. **Configuration du Client Redis** :

- `getRedisClient` : Initialise et renvoie un client Redis avec des paramètres par défaut, établissant ainsi la connexion avec le serveur Redis.

7. **Configuration du Client MongoDB** :
   - `getMongoClientAndCollection` : Configure le client MongoDB et récupère la collection pour assurer l'intégration avec la base de données MongoDB.

Mesure du Temps d'Exécution

Le code inclut une minuterie (`startTime` et `elapsedTime`) pour mesurer le temps d'exécution du gestionnaire `/getPost`. Des mesures temporelles supplémentaires sont ajoutées pour quantifier les avantages du cache Redis et de la base de données MongoDB. Par exemple, mesurez le temps passé sur les hits de cache et les opérations MongoDB par rapport aux misses de cache. Ces mesures supplémentaires aideraient à illustrer de manière plus détaillée les gains de performance obtenus grâce à l'utilisation combinée de la cache et de la base de données MongoDB.

```
Server running on port :3333
==== CACHE MISS ON post-3 ====
==== DB MISS ON post-3 ====
Execution time: 235.16836 milliseconds
==== CACHE HIT ON post-3 ====
Execution time: 0.40460 milliseconds
==== CACHE MISS ON post-3 ====
==== DB HIT ON post-3 ====
Execution time: 3.89799 milliseconds
==== CACHE HIT ON post-3 ====
Execution time: 0.53102 milliseconds
==== CACHE HIT ON post-3 ====
Execution time: 0.97816 milliseconds
==== CACHE MISS ON post-2 ====
==== DB MISS ON post-2 ====
Execution time: 65.27110 milliseconds
==== CACHE HIT ON post-2 ====
Execution time: 0.43765 milliseconds
```