Relatório do Projeto.

Nomes: Ana Julia Matozo Rodrigues: 23027461, Letícia Lima da Silva: 23918691, Murilo Euphrasio Brito: 23028685

- **Objetivo**

Este projeto foi desenvolvido para implementar uma API de gerenciamento de produtos com persistência de dados no MySQL e uso de cache com Redis. O objetivo principal é reduzir o tempo de resposta em consultas de produtos e melhorar o desempenho, utilizando uma abordagem de cache para armazenar resultados frequentemente acessados.

- **Estrutura da Solução Implementada**

A solução está dividida em três principais arquivos:

db.ts: Configura as conexões com os bancos Redis e MySQL, usando variáveis de ambiente para definir as credenciais. Redis é utilizado como cache para melhorar a performance nas consultas, enquanto o MySQL é o banco de dados relacional onde os dados são persistidos.

ProductsRepository.ts: Esta classe gerencia todas as operações CRUD (Create, Read, Update, Delete) dos produtos. Ela implementa a lógica de sincronização entre o MySQL e o Redis. Cada método tenta primeiro buscar ou atualizar os dados no Redis antes de acessar o MySQL, garantindo que o cache esteja sempre atualizado: Método getAll: Busca todos os produtos. Verifica o Redis e, caso os dados não estejam em cache, consulta o MySQL e armazena os resultados no Redis. Método getById: Busca um produto específico pelo ID. Primeiramente, tenta obter o produto do cache e, se não estiver lá, busca no MySQL e armazena no Redis. Método add: Adiciona um novo produto no banco de dados e atualiza o cache invalidando a lista completa de produtos. Método delete: Remove um produto específico do banco e também do cache, invalidando o cache da lista completa para manter a consistência.

server.ts: Configura as rotas da API para realizar operações CRUD nos produtos. Usa a classe ProductsRepository para executar as operações, que realiza a busca e manipulação dos dados entre o MySQL e o Redis. Endpoints: GET /getAllProducts: Retorna a lista completa de produtos. GET /getProduct/ : Retorna um produto específico pelo ID. POST /addProduct: Adiciona um novo produto com base nos dados recebidos no corpo da requisição. DELETE /deleteProduct/ : Remove um produto específico pelo ID.

- **Benefícios da Solução**

Desempenho Acelerado: Utilizando o Redis como cache, reduzimos o número de consultas ao MySQL, o que melhora o tempo de resposta nas consultas mais frequentes. Consistência de Dados: A cada atualização no banco de dados, o cache é atualizado ou invalidado. Isso evita problemas de dados desatualizados em cache. Escalabilidade: Ao reduzir a carga no banco MySQL, a API pode lidar com mais requisições simultâneas sem degradar o desempenho.

- **Problemas que a Solução Não Resolve**

Expiração do Cache: A solução atual não define uma política de expiração para os dados no Redis. Dados cacheados permanecem até serem explicitamente invalidados. Isso pode ser um problema se o sistema crescer significativamente, pois os dados desatualizados podem permanecer em cache indefinidamente.

Concorrência de Atualizações: O cache é atualizado ou invalidado sempre que há mudanças no banco, mas a solução atual não considera situações de alta concorrência, onde múltiplas atualizações podem ocorrer simultaneamente. Nesse caso, podem ocorrer situações onde dados inconsistentes sejam exibidos temporariamente.

Problemas de Conectividade: A solução pressupõe que tanto o Redis quanto o MySQL estão sempre disponíveis. Em caso de falha de conectividade com o Redis, o sistema continuaria a funcionar, mas perderia os benefícios de desempenho do cache. No entanto, a ausência do MySQL resultaria em erro ao tentar acessar ou modificar dados, já que é o repositório de dados primário.

Atualizações Parciais: A implementação atual foca nas operações de criação e exclusão, mas não inclui operações de atualização de produtos. A adição de uma operação update exigiria uma lógica adicional para garantir que o cache seja atualizado apropriadamente.

- **Resultados Observados**

Durante os testes da API, os seguintes comportamentos foram observados:

Redução de Tempo de Resposta: A primeira consulta para getAllProducts e getProduct/:id leva mais tempo, pois os dados são carregados diretamente do MySQL. Consultas subsequentes, no entanto, são atendidas pelo Redis e exibem uma resposta muito mais rápida, evidenciando a eficácia do cache.

Sincronização Consistente: A sincronização entre Redis e MySQL funcionou conforme esperado. Após operações de criação ou exclusão, a invalidação do cache garantiu que novas consultas retornassem dados atualizados diretamente do banco.

Estabilidade Geral: Durante os testes, a aplicação manteve-se estável e funcional, tanto nas operações de cache quanto de acesso ao banco de dados. No entanto, seria recomendado implementar um monitoramento para capturar métricas de desempenho e uso do cache em cenários reais

- **Conclusão**

A solução implementada cumpre os requisitos de gerenciamento de produtos com melhoria de desempenho usando o Redis como cache, garantindo dados consistentes entre cache e banco. Embora a solução apresente limitações, ela oferece uma base sólida para um sistema escalável e eficiente. Com as melhorias sugeridas, a API pode ser ainda mais otimizada para cenários de alta demanda e escalabilidade.