

Entrega 3 - Coordenação, Nomeação e Consistência

Disciplina: Sistemas Distribuídos

Data de Entrega: 31/10/2025

Alunos: Danilo Carvalho De Oliveira, Guilherme Faria da Silva, Vinicius Henrique Domingues

1. Visão Geral da Entrega

Esta entrega foca na implementação de três conceitos fundamentais de sistemas distribuídos, avançando a arquitetura de microsserviços do projeto "Lista de Compras Compartilhada". O objetivo foi implementar mecanismos de coordenação (exclusão mútua), nomeação (descoberta de serviço) e garantir a consistência e replicação dos dados.

2. Coordenação (Exclusão Mútua)

O requisito de coordenação foi aplicado para resolver uma "condição de corrida" (race condition) no `itens-service`.

Problema: Se dois usuários tentarem marcar/desmarcar o mesmo item (evento `marcar_item` no Socket.IO) exatamente ao mesmo tempo, ambos poderiam ler o estado "desmarcado", processar e o último a escrever sobrescreveria a ação do outro, gerando inconsistência.

Solução: Foi implementado um padrão de **Distributed Lock (Trava Distribuída)**, utilizando o MongoDB Atlas como coordenador central.

Implementação:

- Aquisição da Trava (Lock):** Quando um processo recebe o evento `marcar_item`, ele tenta **criar** um novo documento na coleção `Locks` usando o `_id` do item como chave primária (`await Lock.create({ _id: itemId })`).
- Atomicidade:** O MongoDB garante a **atomicidade e unicidade** do campo `_id`. Apenas o **primeiro** processo a executar o `create` consegue criar o documento e, portanto, "adquire a trava".

3. **Deteção de Conflito:** Qualquer outro processo que tente criar um documento com o mesmo `_id` (enquanto a trava estiver ativa) receberá um erro de chave duplicada (`error.code === 11000`). Nosso código captura esse erro e entende que o recurso está "ocupado", impedindo a condição de corrida.
 4. **Liberação da Trava (Unlock):** A atualização do item no banco e a liberação da trava (`await Lock.deleteOne({ _id: itemId })`) são colocadas dentro de um bloco `try...finally`, garantindo que a trava seja **sempre liberada**, mesmo se a atualização falhar.
 5. **Tolerância a Falhas (Prevenção de Deadlock):** Para evitar que uma falha no `itens-service` mantenha um item travado indefinidamente, foi utilizado um **Índice TTL (Time-To-Live)** no Schema do `Lock`. A trava expira automaticamente do banco de dados após 10 segundos (`expires: '10s'`), garantindo a resiliência do sistema.
-

3. Nomeação (Service Discovery)

O requisito de nomeação foi resolvido utilizando o mecanismo de **Service Discovery (Descoberta de Serviço)** nativo do **Docker Compose**.

Problema: Em um ambiente de microsserviços, os contêineres possuem endereços IP internos que são efêmeros (podem mudar a cada reinício). O `proxy` (Nginx) precisa de uma forma confiável de encontrar o `listas-service` e o `itens-service`.

Solução: O `docker-compose.yml` define uma rede interna padrão para todos os serviços.

Implementação: O Docker atua como um servidor DNS para esta rede. Cada serviço é "registrado" automaticamente usando seu nome lógico.

- O `proxy` (Nginx) não precisa saber o IP do contêiner. Ele apenas encaminha as requisições para os nomes lógicos:
 - `http://listas-service:3001` (para a API de Listas)
 - `http://itens-service:3002` (para a API de Itens e WebSockets)
 - Este esquema de nomes lógicos (`listas-service`, `itens-service`) é a implementação do requisito de nomeação.
-

4. Consistência e Replicação

Ambos os requisitos foram atendidos através da escolha da plataforma de banco de dados.





Replicação de Dados:

- **Solução:** Ao utilizar o **MongoDB Atlas**, o requisito de replicação é gerenciado nativamente pela plataforma.
- **Implementação:** O MongoDB Atlas provisiona todos os clusters (incluindo o nível gratuito) como um **Replica Set** (geralmente um cluster com 3 nós). Isso significa que nossos dados são replicados automaticamente em diferentes instâncias, garantindo alta disponibilidade e redundância sem a necessidade de configuração manual da replicação.

Modelo de Consistência:

- **Solução:** O modelo de consistência adotado para a aplicação é o de **Consistência Forte (Strong Consistency)**.
 - **Implementação:** Por padrão, as operações de escrita (`create`, `findByIdAndUpdate`) e leitura (`find`) no MongoDB são direcionadas ao nó **Primário** do Replica Set. Isso garante que, após uma escrita ser confirmada (`await`), qualquer leitura subsequente (como um "refresh" da lista) verá imediatamente o dado mais recente, evitando que o usuário veja dados desatualizados.
-

5. Requisitos Atendidos na Entrega

-  **Coordenação:** Implementado (Distributed Lock com MongoDB e TTL)
-  **Nomeação:** Implementado (Service Discovery via DNS do Docker Compose)
-  **Replicação:** Implementado (Replica Sets nativos do MongoDB Atlas)
-  **Modelo de Consistência:** Definido (Consistência Forte, garantida pelo acesso ao nó Primário)