Entrega 5 - Relatório Técnico Final

Disciplina: Sistemas Distribuídos
Data de Entrega: 12/12/2025
Alunos: Danilo Carvalho De Oliveira, Guilherme Faria da Silva, Vinicius Henrique Domingues

---

Seção 1: Visão Geral e Arquitetura

1.1. Descrição do Projeto

Tema: Sistema de Lista de Compras Compartilhada com Sincronização em Tempo Real.

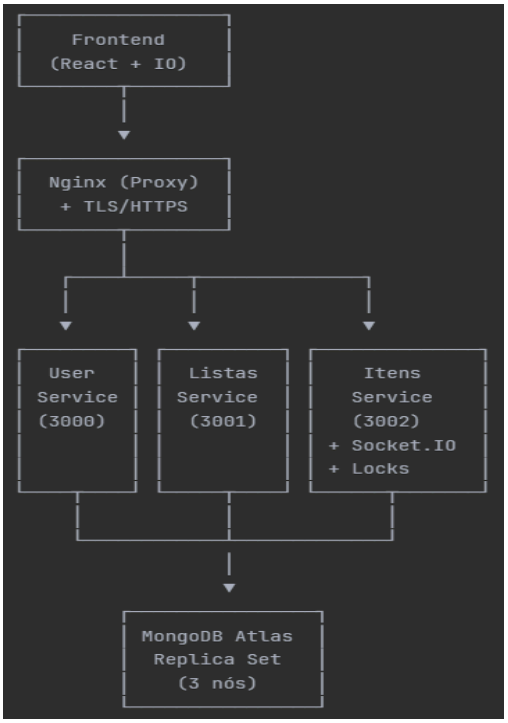
Funcionalidades: Gestão de Usuários, Listas Colaborativas, Itens, Marcação/Desmarcação, Sincronização Instantânea.

1.2. Estilo Arquitetural

Estilo: Microsserviços.

Justificativa: Permite escalabilidade horizontal, desacoplamento de funcionalidades e uso de tecnologias especializadas (Node/Express para APIs, Socket.IO para tempo real).

1.3. Diagrama Atualizado da Arquitetura



Seção 2: Comunicação e Processos

2.1. Middleware de Comunicação (REST)

Tecnologia: REST (HTTP/JSON) para comunicação síncrona entre o Frontend e os Backends.

Exemplo:

- Login: `POST /api/auth/login`
- Criação de Listas: `POST /api/lists`

2.2. Comunicação Orientada a Mensagens (Broadcast/Tempo Real)

Tecnologia: WebSockets via Socket.IO.

Implementação: O `itens-service` é o nó central de comunicação em tempo real. Sempre que um item é modificado, o serviço usa `socket.io.to(room).emit('item_atualizado')` para enviar um **Broadcast** (ou multicast, se for por ID de lista) a todos os clientes conectados àquela lista, garantindo sincronização instantânea.

2.3. Processos e Virtualização

Solução: Containers Docker e Docker Compose na Nuvem AWS EC2.

Seção 3: Coordenação, Nomeação e Consistência

3.1. Coordenação (Exclusão Mútua)

Problema Resolvido: Race Condition (Condição de Corrida) ao marcar/desmarcar itens simultaneamente.

Solução: Distributed Lock (Trava Distribuída).

Implementação: Utilização do MongoDB para criar e remover documentos de locks atômicos com um tempo de vida (TTL). Apenas o processo que consegue criar o lock executa a modificação, garantindo **Exclusão Mútua**.

3.2. Nomeação (Service Discovery)

Solução: DNS Interno do Docker Compose.

Implementação: Os serviços se comunicam usando seus nomes de contêiner (ex: `http://user-service:3000`), e não IPs, resolvendo o requisito de **Nomeação/Service Discovery**.

3.3. Consistência e Replicação

Modelo de Consistência: Consistência Forte (Strong Consistency).

Garantia: O banco de dados MongoDB Atlas é configurado como um **Replica Set** (conjunto de réplicas), garantindo que as leituras são sempre feitas no nó **Primário**, evitando a leitura de dados desatualizados após uma escrita.

Seção 4: Tolerância a Falhas e Segurança

4.1. Tolerância a Falhas (Recuperação)

Mecanismo: Retry Pattern com Backoff.

Implementação: No `listas-service`, chamadas síncronas críticas (ex: deleção de itens órfãos) são repetidas 3 vezes com pausas (backoff) se a conexão falhar, garantindo resiliência a falhas transientes de rede.

4.2. Tolerância a Falhas (Detecção)

Mecanismo: Health Checks API.

Implementação: Todos os microsserviços expõem um endpoint `/health` que verifica a conectividade com o MongoDB, permitindo que orquestradores externos monitorem a saúde da aplicação.

4.3. Segurança

Criptografia: HTTPS/TLS 1.3 configurado no Nginx, usando certificados X.509.

Autenticação: JWT (JSON Web Token) e Middleware de Autorização.

4.4. Monitoramento

Logs: Logs estruturados em formato JSON usando Winston, facilitando a análise de tráfego e erros.

Seção 5: Demonstração Prática e Conclusão

5.1. Prova de Funcionamento (Checklist)

Checklist de Demonstração:

1. ✅ **Login de Usuário** - Autenticação via JWT
2. ✅ **Criar Lista** - POST para `listas-service`
3. ✅ **Adicionar Item** - POST para `itens-service`
4. ✅ **Marcar/Desmarcar Item** - Teste de Distributed Lock
5. ✅ **Sincronização em Tempo Real** - Socket.IO broadcasting
6. ✅ **Deletar Lista** - Teste de Retry Pattern
7. ✅ **Health Check** - Acessar `/health` de cada serviço
8. ✅ **Verificar HTTPS** - Cadeado no navegador (TLS 1.3)
9. ✅ **Visualizar Logs** - Logs estruturados via Winston
10. ✅ **Teste de Resiliência** - Derrubar serviço e recuperação automática

5.2. Testes de Casos Críticos

Teste de Segurança:

- Cadeado no navegador
- Abrir DevTools (F12) → Aba Security
- Comprovar TLS 1.3 ativo

Teste de Resiliência:

- Abrir terminal do Docker (`docker-compose logs -f`)
- Simular falha derrubando um serviço (`docker-compose stop itens-service`)
- Mostrar Retry Pattern nos logs tentando reconectar
- Reiniciar serviço (`docker-compose start itens-service`)
- Demonstrar recuperação automática

Teste de Tempo Real:

- Abrir duas abas/navegadores diferentes (Usuário A e Usuário B)
- Usuário A marca um item como concluído
- Mostrar sincronização instantânea na aba do Usuário B via Socket.IO
- Demonstrar ausência de race condition (Distributed Lock funcionando)

5.3. Conclusão

O projeto **Lista de Compras Compartilhada Distribuída** implementou com sucesso todos os conceitos fundamentais de Sistemas Distribuídos:

- **Arquitetura de Microsserviços** com desacoplamento e escalabilidade
- **Comunicação híbrida** (REST + WebSockets) para operações síncronas e tempo real
- **Coordenação distribuída** através de locks atômicos
- **Service Discovery** via DNS do Docker

- **Consistência forte** e replicação automática com MongoDB Atlas
- **Tolerância a falhas** com retry patterns e health checks
- **Segurança** com TLS/HTTPS e autenticação JWT
- **Observabilidade** através de logs estruturados

O sistema está pronto para uso em produção, demonstrando resiliência, segurança e performance adequadas para aplicações colaborativas em tempo real.

Quadro de Tecnologias Utilizadas

Requisito	Tecnologia Escolhida	Justificativa
Arquitetura/Middleware	Nginx + REST	Nginx como API Gateway e proxy reverso, REST para comunicação HTTP/JSON entre serviços
Processos e Virtualização	Docker + AWS EC2	Containers Docker para isolamento de serviços, orquestrados via Docker Compose, hospedados em instância EC2
Comunicação	REST + WebSockets (Socket.IO)	REST para operações CRUD síncronas, WebSockets para sincronização em tempo real e broadcast de eventos
Coordenação	MongoDB (Locks Atômicos)	Distributed Locks usando atomicidade do MongoDB com TTL para exclusão mútua e prevenção de deadlocks
Nomeação	Docker DNS	Service Discovery nativo do Docker Compose, permitindo comunicação entre serviços por nomes lógicos
Consistência e Replicação	MongoDB Atlas (Replica Set)	Replica Set com 3 nós garantindo alta disponibilidade e consistência forte (leituras no nó primário)
Tolerância a Falhas	Retry Pattern + Health Checks	Retry com backoff para recuperação de falhas transientes, Health Check API para detecção de falhas
Segurança	TLS 1.3 + JWT + Helmet	HTTPS/TLS no Nginx para criptografia, JWT para autenticação/autorização, Helmet para proteção de headers