

**Sistemas Distribuídos**  
Trabalho Prático Final  
Sistema de Chat Distribuído

Letícia de Oliveira Soares  
Mateus Gonçalves Soares

## 1. Introdução

Este trabalho apresenta o desenvolvimento de um sistema de chat distribuído em tempo real, fundamentado em conceitos centrais de Sistemas Distribuídos, como microsserviços, comunicação assíncrona, persistência de dados e escalabilidade horizontal. Sistemas de mensagens instantâneas exigem baixa latência, suporte a múltiplos usuários simultâneos e alta disponibilidade, o que os torna um cenário adequado para aplicação prática desses conceitos.

A arquitetura do sistema é composta por dois microsserviços independentes, auth-service e chat-service, com autenticação baseada em JWT, permitindo operação stateless e fácil replicação. A comunicação em tempo real é realizada via WebSockets (Socket.IO) e a escalabilidade horizontal é garantida pelo uso do Redis como message broker, possibilitando a sincronização entre múltiplas instâncias do serviço de mensagens. O sistema utiliza persistência de dados e é validado por testes unitários, de integração e de concorrência/carga, comprovando o atendimento aos requisitos propostos.

**Repositório:** <https://github.com/leticiaasoares/sd-chat/tree/main>

## 2. Arquitetura do Sistema

A arquitetura do sistema é composta por três camadas principais: Auth-Service (Serviço de Autenticação e Usuários), Chat-Service (Serviço de Mensagens em Tempo Real) e Front-end Web (Cliente).

Além disso, o sistema utiliza dois componentes de infraestrutura: banco de dados SQLite para persistência de usuários e mensagens e Redis como middleware de comunicação entre instâncias do chat-service.

### Versão simplificada da arquitetura:



### 3. Microsserviços e Separação de Responsabilidades

#### Auth-Service

O auth-service é responsável por: cadastro de novos usuários, autenticação via login e senha, geração de tokens JWT e listagem de usuários cadastrados.

As características principais são serviço stateless (não mantém sessão em memória), autenticação baseada em JWT (JSON Web Token) e permitir múltiplas instâncias simultâneas sem necessidade de sincronização de estado.

#### Chat-Service

O chat-service é responsável por: comunicação em tempo real via WebSockets (Socket.IO), envio e recebimento de mensagens privadas, persistência de mensagens no banco de dados e sincronização de mensagens entre múltiplas instâncias usando Redis.

Cada instância do chat-service valida o usuário via JWT, insere o usuário em uma sala identificada pelo seu username e utiliza Redis para propagar mensagens entre instâncias.

### 4. Comunicação Assíncrona e Tempo Real

A comunicação em tempo real é implementada utilizando WebSockets, por meio da biblioteca Socket.IO.

O cliente estabelece uma conexão persistente com o chat-service, então as mensagens são enviadas e recebidas por eventos assíncronos. Não há necessidade de polling para mensagens e o servidor envia dados ao cliente assim que eventos ocorrem.

Para a integração com Redis, o Redis Adapter do Socket.IO é utilizado. As mensagens enviadas em uma instância do chat-service são automaticamente propagadas para as demais e os usuários recebem mensagens mesmo estando conectados a instâncias diferentes.

### 5. Persistência de Dados

O sistema utiliza um banco de dados SQLite, contendo duas tabelas principais:

- users: armazena informações de usuários e credenciais (com senha criptografada);
- messages: armazena mensagens trocadas entre usuários, com remetente, destinatário, conteúdo e timestamp.

A persistência de dados é independente da conexão do usuário, o histórico de mensagens é acessível após logout ou reconexão e o banco compartilhado é entre instâncias.

### 6. Execução

#### Iniciar o Redis:

O Redis deve estar em execução antes de iniciar o chat-service.

Verifique se o Redis está ativo:

```
redis-cli ping
```

#### Rodar o Auth-Service

Em um terminal:

```
cd auth-service
npm install
npm start
```

O serviço de autenticação ficará disponível em: <http://localhost:4000>

### Rodar o Chat-Service (Escalabilidade Horizontal)

O chat-service pode ser executado em múltiplas instâncias.

Cada instância deve ser iniciada em um terminal diferente, usando portas distintas.

Exemplo com três instâncias:

```
cd chat-service
npm install
```

```
PORT=4001 REDIS_URL=redis://localhost:6379 node server.js
```

```
PORT=4002 REDIS_URL=redis://localhost:6379 node server.js
```

```
PORT=4003 REDIS_URL=redis://localhost:6379 node server.js
```

### Acessar o Front-end

O front-end é servido pelo próprio chat-service.

Acesse pelo navegador:

<http://localhost:4001>

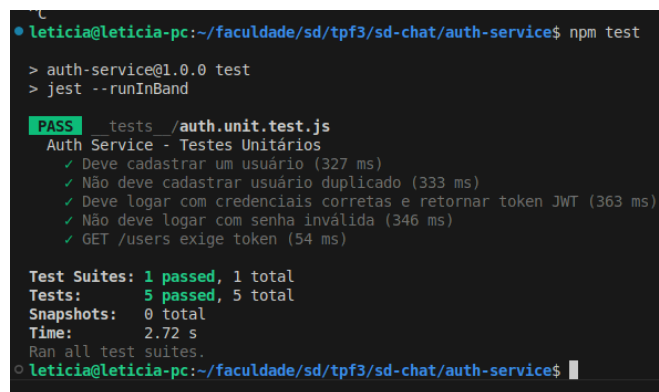
<http://localhost:4002>

Cada endereço corresponde a uma instância diferente do chat-service.

## 7. Testes

### Testes Unitários

No serviço de autenticação (auth-service), os testes unitários cobrem os principais cenários relacionados ao gerenciamento de usuários. Foram testados o cadastro de novos usuários, a validação de credenciais durante o processo de login, a geração correta de tokens JWT e o comportamento do sistema diante de tentativas de autenticação inválidas, como senhas incorretas ou usuários duplicados. Também foi verificada a proteção de rotas que exigem autenticação, garantindo que apenas usuários autenticados tenham acesso a determinadas funcionalidades. Os testes estão em `auth-service/__tests__/auth.unit.test.js`.



```
leticia@leticia-pc:~/faculdade/sd/tpf3/sd-chat/auth-service$ npm test
> auth-service@1.0.0 test
> jest --runInBand

PASS __tests__/auth.unit.test.js
Auth Service - Testes Unitários
  ✓ Deve cadastrar um usuário (327 ms)
  ✓ Não deve cadastrar usuário duplicado (333 ms)
  ✓ Deve logar com credenciais corretas e retornar token JWT (363 ms)
  ✓ Não deve logar com senha inválida (346 ms)
  ✓ GET /users exige token (54 ms)

Test Suites: 1 passed, 1 total
Tests: 5 passed, 5 total
Snapshots: 0 total
Time: 2.72 s
Ran all test suites.
leticia@leticia-pc:~/faculdade/sd/tpf3/sd-chat/auth-service$
```

Resultado dos testes do serviço de autenticação

No serviço de mensagens (chat-service), os testes unitários concentram-se na validação da persistência e recuperação das mensagens. Foram testados o armazenamento correto das mensagens no banco de dados e a recuperação do histórico de conversas entre dois usuários autenticados. Esses testes asseguram que o serviço de mensagens mantenha a integridade dos dados. Os testes estão em chat-service/\_\_\_tests\_\_\_/chat.unit.test.js.

```
leticia@leticia-pc:~/faculdade/sd/tpf3/sd-chat/chat-service$ npm test
> chat-service@1.0.0 test
> jest --runInBand

console.log
  Chat service rodando na porta 4001 (Redis em redis://localhost:6379)

    at Server.log (server.js:143:15)

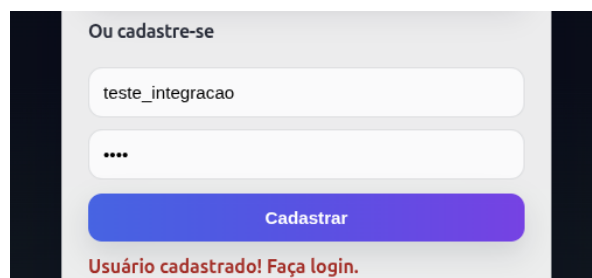
PASS tests /chat.unit.test.js
  Chat Service - Testes Unitários (Persistência)
    ✓ Deve persistir mensagens e recuperar histórico corretamente (229 ms)
    ✓ Endpoint /messages exige token (49 ms)

Test Suites: 1 passed, 1 total
Tests: 2 passed, 2 total
Snapshots: 0 total
Time: 2.185 s
Ran all test suites.
```

Resultado dos testes do serviço de mensagens

### Testes de Integração

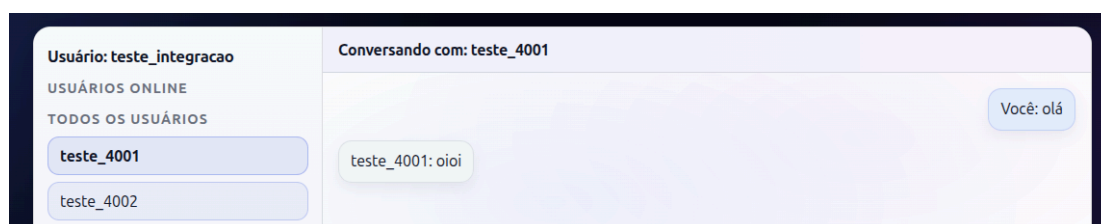
O front-end foi testado e a comunicação entre o serviço de autenticação e o serviço de mensagens ocorreu corretamente. Um usuário foi cadastrado, autenticado e em seguida enviou uma mensagem:



Esta imagem mostra a interface de usuário para o cadastro. No topo, há o título "Ou cadastre-se". Abaixo dele, há dois campos de entrada: um para o nome de usuário, contendo "teste\_integracao", e outro para a senha, contendo "....". Abaixo dos campos, há um botão azul com o texto "Cadastrar". Na base da tela, há uma mensagem de feedback em vermelho: "Usuário cadastrado! Faça login."



Esta imagem mostra a interface de usuário para o login. No topo, há o título "Login". Abaixo dele, há dois campos de entrada: um para o nome de usuário, contendo "teste\_integracao", e outro para a senha, contendo "....". Abaixo dos campos, há um botão azul com o texto "Entrar".



Esta imagem mostra a interface de usuário para o chat. No topo, há uma barra de status que indica "Usuário: teste\_integracao" e "Conversando com: teste\_4001". Abaixo disso, há uma lista de usuários online sob o título "USUÁRIOS ONLINE" e "TODOS OS USUÁRIOS". A lista contém dois nomes: "teste\_4001" e "teste\_4002". À direita, há uma caixa de texto para enviar mensagens, com o placeholder "Você: olá". Abaixo da caixa de texto, há uma mensagem recebida: "teste\_4001: oi oi".

## Testes de Concorrência/Carga

Durante a execução, os usuários u1 a u10 foram inicialmente cadastrados e autenticados no auth-service, obtendo tokens JWT válidos. Em seguida, cada usuário estabeleceu uma conexão WebSocket com uma instância específica do chat-service. A distribuição ocorreu entre três instâncias diferentes (http://localhost:4001, http://localhost:4002 e http://localhost:4003), garantindo que o teste representasse um ambiente horizontalmente escalado. Esse passo valida que usuários conectados a instâncias distintas ainda conseguem se comunicar corretamente.

Após as conexões serem estabelecidas, foi realizado o envio de mensagens simultâneas em formato de ciclo (cada usuário enviando uma mensagem para o próximo usuário, totalizando 10 mensagens). Os resultados indicaram que cada usuário recebeu exatamente uma mensagem, e o total de mensagens entregues foi 10, igual ao esperado (Total entregue: 10 (esperado: 10)). Esse resultado comprova que o sistema manteve a consistência da entrega e operou corretamente sob concorrência, demonstrando que o uso do Redis como mecanismo de sincronização entre instâncias do chat-service permite comunicação em tempo real em um ambiente distribuído, validando a escalabilidade horizontal na prática.

```
leticia@leticia-pc:~/faculdade/sd/tpf3/sd-chat$ node tests/load_test_10users.js
== Preparando usuários ==
== Login e tokens ==
Distribuição nas instâncias:
- u1 -> http://localhost:4002
- u2 -> http://localhost:4003
- u3 -> http://localhost:4001
- u4 -> http://localhost:4002
- u5 -> http://localhost:4003
- u6 -> http://localhost:4001
- u7 -> http://localhost:4002
- u8 -> http://localhost:4003
- u9 -> http://localhost:4001
- u10 -> http://localhost:4002
== Conectando sockets ==
== Enviando mensagens simultâneas (carga) ==
== Resultado ==
u1 recebeu 1 mensagens
u2 recebeu 1 mensagens
u3 recebeu 1 mensagens
u4 recebeu 1 mensagens
u5 recebeu 1 mensagens
u6 recebeu 1 mensagens
u7 recebeu 1 mensagens
u8 recebeu 1 mensagens
u9 recebeu 1 mensagens
u10 recebeu 1 mensagens
Total entregue: 10 (esperado: 10)
✅ Teste de carga OK: mensagens entregues com múltiplas instâncias (escalabilidade horizontal comprovada)
leticia@leticia-pc:~/faculdade/sd/tpf3/sd-chat$
```