Sistemas Distribuídos
Planejamento de uma aplicação distribuída

Thiago O. da Silva[1], **Rhuan M. O. M. de Carvalho**[2]

[1]*Departamento de Ciência da Computação – Universidade Federal do Piauí (UFPI)*
Teresina – PI – Brazil

[2]*Departamento de Ciência da Computação – Universidade Federal do Piauí (UFPI)*
Teresina – PI – Brazil Teresina – PI – Brazil

{thiago_silva,rhuanmourao}@ufpi.edu.br

Abstract. *This report aims to plan a distributed system application, considering the architectural styles of both the client and server parts.*

Resumo. *Este relatório tem como objetivo planejar uma aplicação de um sistema distribuído, levando em consideração os estilos arquitetônicos tanto da parte cliente quanto da parte servidora.*

1. Domínio de aplicação

No contexto do trabalho, optou-se por desenvolver um sistema de chat em tempo real utilizando WebSockets com React e Node.js. Com isso, essa escolha permite explorar a comunicação e interação entre o cliente e o servidor, onde o servidor WebSocket gerencia as conexões dos clientes, roteando as mensagens, e o cliente React exibe a interface do chat, permitindo que os usuários enviem e recebam mensagens instantaneamente.

2. Requisitos

2.1. Requisitos Funcionais:

- Registro de Usuário: Permitir que os usuários se registrem na aplicação fornecendo um nome de usuário e senha.
- Autenticação: Autenticar os usuários registrados para garantir o acesso seguro ao chat em tempo real;
- Envio de Mensagens: Permitir que os usuários enviem mensagens em tempo real para outros participantes do chat.
- Recebimento de Mensagens: Exibir as mensagens recebidas instantaneamente na interface do usuário.
- Lista de Contatos: Apresentar uma lista de contatos disponíveis para iniciar conversas individuais ou em grupo.
- Notificações: Notificar os usuários sobre novas mensagens recebidas ou atividades relevantes no chat.

2.2. Requisitos Não Funcionais:

- Escalabilidade: O sistema deve ser capaz de lidar com um número crescente de usuários e conexões simultâneas, garantindo um desempenho adequado em cenários de alta carga.

- Disponibilidade: A aplicação deve estar disponível de forma contínua, minimizando possíveis períodos de interrupção ou manutenção planejada.
- Segurança: As comunicações devem ser protegidas por meio de criptografia, garantindo a confidencialidade e a integridade das mensagens transmitidas.
- Desempenho: A aplicação deve apresentar um desempenho responsivo, com baixa latência na entrega de mensagens e uma interface ágil e fluida para uma experiência do usuário satisfatória.

3. Planejando a arquitetura da parte cliente

Levando em consideração a necessidade de uma aplicação responsiva e de baixa latência, que garanta uma experiência de chat em tempo real fluida para os usuários, optou-se por uma abordagem baseada em arquitetura cliente leve.

Com essa arquitetura, podemos concentrar a maior parte do processamento e lógica no servidor, enquanto o cliente é responsável principalmente pela exibição da interface de usuário e pela interação com o servidor por meio de requisições WebSocket. Dessa forma, isso permite reduzir a carga de trabalho no cliente e aproveitar o poder de processamento e recursos do servidor.

Ademais, o React como framework de desenvolvimento do cliente, foi um ponto importante de decisão, já que proporciona uma renderização rápida e eficiente da interface do usuário através da sua abordagem de componentização.

3.1. Benefícios:

- Modularidade e reutilização de componentes.
- Organização clara e escalável.
- Melhor aproveitamento dos recursos do framework React.

3.2. Desafios:

- Gerenciamento do estado compartilhado entre os componentes.
- Manutenção à medida que cresce em complexidade.
- Necessidade de projetar componentes altamente modulares e independentes.

4. Planejando a arquitetura da parte servidora

Optou-se por utilizar o Node.js como plataforma de desenvolvimento do servidor devido à sua capacidade de lidar com conexões de entrada e saída assíncronas de forma eficiente. Além disso, foi planejada uma arquitetura modular, dividindo as responsabilidades em diferentes componentes, como:

- Gerenciamento de Conexões: Utiliza-se a biblioteca Socket.IO no lado do servidor para estabelecer conexões WebSocket com os clientes, usando métodos próprios para receber e enviar mensagens, como o ”io.emit”.
- Lógica do Chat: Como o Socket.IO oferece uma ampla gama de recursos e flexibilidade, será implementando um módulo no servidor para lidar com a lógica do chat, permitindo aplicar métodos de validação e filtro de mensanges, gerenciamento de salas e moderação, conforme necessário.
- Persistência de Dados: Será utilizado um banco de dados para armazenar as mensagens do chat e outras informações relevantes, como detalhes do usuário.

4.1. Benefícios:

- Facilidade de teste e depuração.
- Separação clara de responsabilidades entre os diferentes módulos.
- Escalabilidade eficiente ao adicionar ou remover instâncias do servidor conforme necessário.

4.2. Desafios:

- Gerenciamento das dependências entre os módulos.
- Sincronização eficiente dos dados em tempo real entre os diferentes módulos envolvidos no chat.
- Implementação de medidas de segurança.

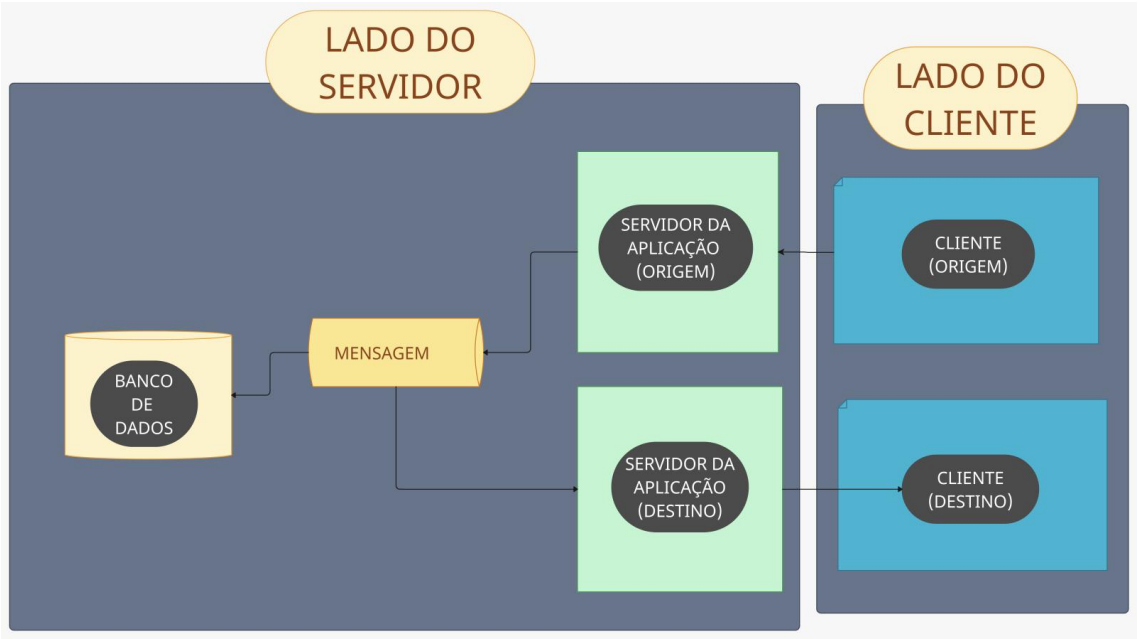

Figure 1. Arquitetura