Relatório de Implementação do Servidor e Conexão com o Cliente

1. Introdução

Este relatório detalha as principais escolhas de implementação para o servidor Flask e sua comunicação com o cliente JavaScript. O sistema é um chat baseado em salas, permitindo que usuários entrem, saiam e enviem mensagens de forma dinâmica. A comunicação entre cliente e servidor ocorre por meio de uma API RESTful.

---

2. Escolhas de Implementação no Servidor

2.1 Uso do Flask

O Flask foi escolhido por ser um microframework leve e flexível para a criação de APIs RESTful. Ele permite a rápida implementação de endpoints HTTP e facilita o gerenciamento de requisições e respostas JSON.

2.2 Estrutura de Armazenamento de Dados

O servidor usa **dicionários Python** para armazenar informações sobre salas e mensagens:

- `rooms`: Um dicionário onde as chaves são nomes de salas e os valores são conjuntos de usuários ativos.
- `room_messages`: Um dicionário que armazena histórico de mensagens para cada sala, limitando a 50 mensagens para evitar sobrecarga de memória.

2.3 Gerenciamento de Salas

O servidor permite que usuários criem e entrem em salas de chat dinamicamente. Quando o primeiro usuário entra, a sala é criada. Se uma sala fica vazia, ela é automaticamente removida para liberar recursos.

2.4 Envio e Recebimento de Mensagens

- **Mensagens são armazenadas no servidor** e retornadas ao cliente quando solicitado.
- Um mecanismo de **timestamp** é usado para que os clientes busquem apenas mensagens novas, reduzindo o tráfego de rede.

2.5 Uso de CORS

O Flask-CORS foi utilizado para permitir requisições do navegador para o servidor, habilitando a comunicação entre domínios diferentes (Cross-Origin Resource Sharing).

2.6 Serviço de Arquivos Estáticos

O servidor também pode fornecer arquivos estáticos (HTML, CSS, JS) para o cliente, permitindo que a interface do chat seja carregada diretamente do servidor.

---

3. Implementação da Conexão com o Cliente

3.1 Comunicação via REST API

O cliente se comunica com o servidor por meio de requisições HTTP GET e POST para as seguintes rotas:

- `POST /join` - Adiciona um usuário a uma sala.
- `POST /leave` - Remove um usuário de uma sala.
- `POST /send` - Envia uma mensagem para uma sala.
- `GET /receive` - Obtém mensagens novas de uma sala.
- `GET /user_rooms` - Retorna as salas em que o usuário está presente.

3.2 Atualização Contínua do Chat

O cliente faz **requisições periódicas** ao servidor para buscar mensagens novas e atualizar a lista de salas. Isso é feito por meio de:

- `setTimeout(receiveMessages, 1000)`: Busca mensagens novas a cada segundo.
- `setInterval(updateRoomList, 5000)`: Atualiza a lista de salas a cada 5 segundos.

3.3 Interface do Usuário

A interface do chat é manipulada dinamicamente pelo JavaScript, exibindo mensagens e mudanças de sala em tempo real. O cliente adiciona mensagens recebidas na tela e destaca a sala ativa.

---

4. Considerações Finais

O sistema foi projetado para ser leve e eficiente, evitando o uso de banco de dados no backend e adotando uma estrutura baseada em dicionários. Esse modelo é adequado para testes e pequenos grupos, mas pode ser expandido com WebSockets para comunicação em tempo real ou integrado a um banco de dados para persistência de mensagens.

Possíveis melhorias futuras incluem:

- Uso de **WebSockets** para mensagens em tempo real.
- Implementação de autenticação de usuários.
- Armazenamento das mensagens em um banco de dados para persistência.

O projeto atual já demonstra a viabilidade de um chat simples via REST API, funcionando de forma eficaz para troca de mensagens entre usuários em salas virtuais.