Sistemas Distribuídos - GSI529

Sistema de gerenciamento de um restaurante

Nicolly Ribeiro Luz
Matrícula: 32221BSI022

1 Visão geral da implementação

Anteriormente, meu projeto se encontrava em seu estágio inicial, de forma bem simples, a comunicação entre as entidades Cliente e Servidor do Restaurante estava funcionando e havia sido implementada com a utilização de um Socket TCP Multithreading, para ver múltiplos clientes interagindo com o servidor.

Agora, nessa etapa, evolui a comunicação e deixei de usar os sockets anteriormente implementados. O meu sistema evoluiu de uma comunicação básica para um sistema de mensagens robusto que utiliza múltiplos protocolos, cada um adequado a uma finalidade específica, como mensagens estruturadas em formato JSON, protocolo HTTP, e uma API REST externa.

Aqui, a grande novidade é que criei a minhas próprias APIs de forma local, e para a sua construção, utilizei a ferramenta node-red.

Criei duas APIs, uma para o cardápio, que será exibida para o cliente assim que ele se conectar ao servidor do restaurante, e outra que controla todos os pedidos de todos os clientes e gerencia seu status desde o pedido novo/confirmado, até que o pedido seja entregue ao cliente.

1.1 APIs

Fluxos das APIs, no node-red, de forma local:

API Cardápio:

A API do cardápio é responsável por armazenar todos os itens e exibi-los para o cliente, por isso, só conta com métodos GET e POST. **URL:** `http://127.0.0.1:1880/cardapioRest`.

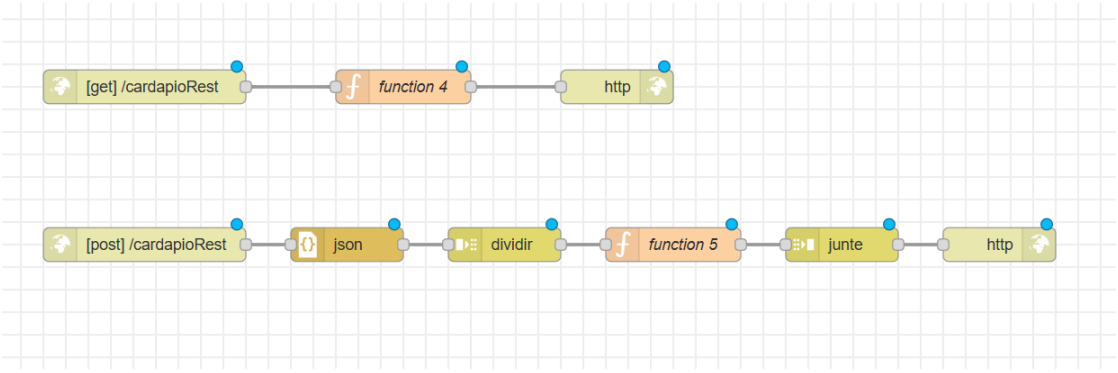

Figura 1: Diagrama de fluxo das rotas da API de cardápio.

API Pedidos:

A API dos Pedidos é responsável por armazenar todos os pedidos de todos os clientes e servir de base para a estação de trabalho que aqui chamaremos de cozinha, e também para o garçom, para isso, temos os métodos GET, POST e PUT, pois os dados dos pedidos serão alterados, principalmente o status.

URL: `http://127.0.0.1:1880/pedidos`.

Para o PUT: `http://127.0.0.1:1880/pedidos/idPedido`, exemplo: `http://127.0.0.1:1880/pedidos/123`.

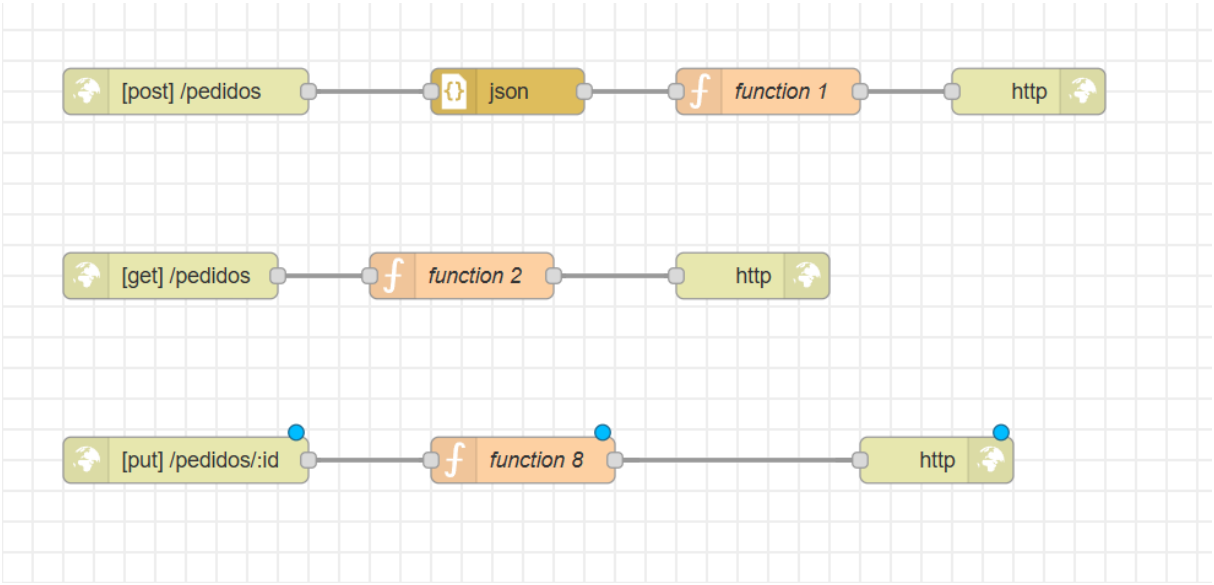

Figura 2: Diagrama de fluxo das rotas da API de pedidos.

2 Projeto de arquitetura de mensagem/fluxo:

O fluxo de dados e de informações que meu sistema está seguindo atualmente é:

1. O Servidor é iniciado.
2. O cliente se conecta com o servidor e ele é encarregado de exibir para o cliente o cardápio vindo da API.
3. O cliente faz o pedido e manda para o servidor.
4. O servidor envia o pedido para a API de pedidos.
5. A cozinha, que fica escutando da API de pedidos, prepara os pedidos assim que eles chegam, mas com uma ressalva, a cozinha funciona como uma fila que só produz 3 pedidos simultaneamente, logo, quando um pedido chega e a cozinha está cheia, ele é enviado para uma lista de espera.
6. Depois de um *timeOut*, (O verdadeiro *timeOut* deveria ser o tempo de preparo de cada prato, mas para uma abordagem mais didática e mais rápida para os testes, usei um time de 1-2 minutos.)

7. Após o *timeOut*, o status do pedido na API de pedidos é alterada para "Pronto".
8. No sistema também há a implementação de uma nova entidade, o **Garçom**, ele fica observando a API de pedidos e toda vez que o status de um pedido é mudada para "Pronto", ele pega o pedido e envia para o cliente e muda o status para "Entregue".

Resumindo, o fluxo de mensagens do sistema é dado por:

Cliente → Servidor → API de Pedidos → Cozinha → API de Pedidos → Garçom → Cliente

2.0.1 Fluxo Contínuo de Dados

O projeto aborda a noção de fluxo contínuo de dados através de um modelo de Polling (sondagem). Os componentes Cozinha e Garcom não esperam ser notificados ativamente; em vez disso, eles consultam a API externa em intervalos regulares (Thread.sleep) para verificar se há novos dados (pedidos "Confirmados"ou "Prontos"). Este é um mecanismo simples e eficaz para simular um sistema reativo a eventos, onde os componentes observam um estado compartilhado (a API de pedidos) e reagem às mudanças que lhes interessam.

2.1 Comandos e eventos

2.1.1 Comandos:

Como comandos podemos citar no meu projeto:

1. **registrarServico:** O ServidorRestaurante ordena ao NameServer que armazene suas informações.
2. **buscarServico:** O ClienteRestaurante solicita ao NameServer o endereço de um serviço.
3. **enviarPedido:** O ClienteRestaurante instrui o ServidorRestaurante a processar um novo pedido.

2.1.2 Eventos:

Como eventos podemos citar:

1. **Atualização de Status (Confirmado -> Preparando -> Pronto -> Entregue):**

   Quando a Cozinha muda o status de um pedido para "Pronto", ela não está ordenando que o Garcom faça algo diretamente. Ela está simplesmente anunciando um fato: "O pedido X está pronto". O Garcom, que está observando esses eventos, reage a essa informação e inicia o processo de entrega. Portanto, a atualização de status na API funciona como a publicação de um evento.

2. **Cozinha observando a API de pedidos:**

   Além disso, a cozinha está sempre escutando na API de pedidos e fica alerta assim que novos pedidos são inseridos na API pelo servidor, assim que um pedido é inserido, ela inicia o tratamento dele de acordo com a disponibilidade da fila de processos que comporta no maximo 3 ao mesmo tempo.

3 Estrutura e Formato das Mensagens-Chave

O sistema utiliza diferentes formatos de mensagem, adaptados ao protocolo de comunicação e à finalidade da interação. Abaixo estão as estruturas das mensagens mais importantes.

3.1 Mensagem de Registro de Serviço

Esta mensagem é um **Comando** enviado ao NameServer para que ele armazene a localização de um serviço.

Protocolo: JSON sobre TCP
Formato: A mensagem é um objeto JSON com dois campos principais: `acao` para definir a intenção e `servico` contendo os dados do serviço.
Estrutura:

```
{
  "acao": "registrar",
  "servico": {
    "nome": "servidor do restaurante",
    "host": "localhost",
    "porta": 8080
  }
}
```

3.2 Mensagem de Busca de Serviço

Este é um **Comando** enviado ao NameServer para descobrir o endereço de um serviço.

Protocolo: JSON sobre TCP
Formato: Um objeto JSON simples contendo a `acao` de busca e o `nomeServico` desejado.
Estrutura:

```
{
  "acao": "buscar",
  "nomeServico": "servidor do restaurante"
}
```

3.3 Mensagem de Envio de Pedido

Este é um **Comando** enviado pelo ClienteRestaurante ao ServidorRestaurante para criar um novo pedido.

Protocolo: HTTP POST
Formato: O corpo (body) da requisição é uma string em Texto Plano (`text/plain`). O formato é uma lista de itens separados por vírgula, onde cada item é representado por ID-QUANTIDADE.
Estrutura (Exemplo):

```
{2-1,3-2}
```

(Significa: 1 unidade do item com ID 2 e 2 unidades do item com ID 3).

3.4 Mensagem de Atualização de Status

Esta mensagem funciona como um **Evento** publicado na API externa pela Cozinha ou pelo Garçom para anunciar uma mudança no estado de um pedido.

Protocolo: HTTP PUT
Formato: O corpo da requisição é o objeto Pedido completo, serializado em JSON (`application/json`), com o campo `status` atualizado.
Estrutura (Exemplo):

```
{
  "idPedido": 5,
  "itens": [
    { "id": 2, "nomeItem": "Hamburguer",
    "quantidade": 1, "precoUnitario": 28.00 }
  ],
  "clienteHost": "127.0.0.1",
  "portaCliente": 8083,
  "status": "Pronto",
  "valorTotal": 28.00
}
```

4 Nomeclatura de processos

Nesta etapa do projeto, foi implementado um mecanismo de descoberta de serviços através de um Servidor de Nomes (NameServer) centralizado. Este novo componente se integra ao sistema desacoplando o ClienteRestaurante do ServidorRestaurante.

Anteriormente, o Cliente precisava ter o endereço (host:porta) do Servidor fixo em seu código. Agora, o Servidor se registra com um nome lógico (ex: "servidor do restaurante") no NameServer ao iniciar. O Cliente, por sua vez, consulta o NameServer para descobrir o endereço atual do Servidor antes de se comunicar.

Essa implementação aumenta a robustez e a flexibilidade do sistema, permitindo que o endereço do ServidorRestaurante seja alterado sem a necessidade de modificar e recompilar o código dos clientes. Os demais componentes, Cozinha e Garcom, continuam operando de forma independente, monitorando o estado dos pedidos através da API externa.

Eu implementei esse mecanismo apenas entre Servidor-Cliente, pois é o cliente que precisa se conectar ao servidor, e ele se conecta apenas com o Servidor, pois toda a comunicação restante é independente e se relacionam principalmente com as APIs descritas acima.

O cliente se conecta ao servidor e todo o resto do fluxo é organizado de forma desacoplada e assíncrona.

5 Guia de Execução do Projeto

Pré-requisitos essenciais

1. **Java Development Kit (JDK) e Java Virtual Machine (JVM) configurados:** Certifique-se de ter o JDK instalado e o ambiente configurado corretamente em seu sistema.
2. **Obtenção dos projetos:** Baixe os projetos do Cliente e do Servidor da pasta do GitHub onde os códigos estão publicados. Acesse o repositório através do link: Link para o Repositório do Projeto
3. **API externa e a ferramenta Node-RED:** A API deve estar em execução e acessível em: `http://127.0.0.1:1880`. (Os fluxos das APIs também estão disponíveis no GitHub descrito acima e devem ser importados no Node-RED.)

   3.1. **Instalação do Node-RED:**

   3.1.1. Instale o Node.js (versão LTS) do site oficial: `https://nodejs.org/`.

   3.1.2. Abra o terminal e instale o Node-RED globalmente:

   ```
   npm install -g --unsafe-perm node-red
   ```

   3.1.3. Inicie o Node-RED com:

   ```
   node-red
   ```

Execução do Projeto

Agora que os projetos já foram baixados: Para compilar e testar o projeto, siga os passos abaixo na ordem correta. É essencial iniciar os componentes em terminais ou consoles separados:

1. **Iniciar o Servidor de Nomes**
2. **Iniciar o Servidor do Restaurante**
3. **Iniciar a Cozinha**
4. **Iniciar o Garçom**
5. **Iniciar um ou mais Clientes, de acordo com sua IDE**

Testes do Sistema

Faça alguns testes para garantir que tudo esteja funcionando corretamente:

1. Faça um pedido no terminal do cliente.
2. Veja no terminal do servidor o ID do pedido gerado.

3. Verifique na API (navegador ou ferramenta como Postman/Insomnia) se o pedido foi inserido na lista de pedidos.
4. Observe na cozinha se o pedido já começou a ser preparado ou se ainda está na fila.
5. Verifique se o status do pedido mudou para “Pronto”.
6. Confirme no terminal do Garçom se o pedido foi entregue ao respectivo cliente.