Universidade Federal do Ceará – Campus Quixadá

Disciplina: Sistemas Distribuídos
Professor: Rafael Braga
Curso: Ciência da Computação
Trabalho: Trabalho 3 – Web Services (WS) / API

Autores: Nícolas Ferreira Leite - 552425 / Patrick de Farias Ramos - 556711

---

1. Introdução

Este trabalho tem como objetivo reimplementar o serviço remoto desenvolvido no Trabalho 2 utilizando uma arquitetura cliente-servidor baseada em Web Services (WS) ou Application Programming Interface (API). A comunicação entre cliente e servidor ocorre por meio do protocolo HTTP, seguindo o modelo de requisição e resposta, sem o uso de sockets ou RMI, conforme especificado pelo enunciado.

A aplicação desenvolvida simula um sistema de farmácia, permitindo o cadastro de produtos, consulta de estoque e realização de vendas, utilizando conceitos de orientação a objetos e serviços distribuídos.

---

2. Tecnologias Utilizadas

- **Linguagem do Servidor:** Python
- **Framework Web:** Flask
- **Linguagem do Cliente:** JavaScript (Node.js)
- **Biblioteca HTTP (Cliente):** Axios
- **Formato de Comunicação:** JSON sobre HTTP

---

3. Arquitetura do Sistema

O sistema é composto por dois componentes principais:

- **Servidor (API REST):** Responsável pela lógica de negócio, armazenamento em memória e disponibilização dos serviços remotos.

- **Cliente:** Responsável por consumir os serviços da API por meio de requisições HTTP.

A arquitetura segue o modelo cliente-servidor, onde o cliente realiza chamadas HTTP (GET e POST) para a API, que processa as requisições e retorna respostas no formato JSON.

---

4. Modelagem Orientada a Objetos

4.1 Herança ("é-um")

- **Produto:** Classe base contendo atributos comuns (`id`, `nome`, `preco`).
- **Medicamento:** Especialização de Produto, adicionando o atributo `tarja`.
- **Perfumaria:** Especialização de Produto, adicionando o atributo `fragrancia`.

4.2 Agregação ("tem-um")

- **Venda:** Possui um objeto do tipo `Cliente` e uma lista de objetos do tipo `Produto`.
- **Cliente:** Representa o comprador, contendo `cpf` e `nome`.

Essa modelagem garante reutilização de código, organização e clareza na estrutura do sistema.

---

5. Serviços Remotos Implementados

5.1 Cadastro de Medicamento

- **Rota:** `POST /medicamentos`
- **Descrição:** Cadastra um novo medicamento no estoque da farmácia.
- **Entrada:** JSON contendo `id`, `nome`, `preco` e `tarja`.
- **Saída:** Confirmação de sucesso ou mensagem de erro.

5.2 Cadastro de Perfumaria

- **Rota:** `POST /perfumarias`
- **Descrição:** Cadastra um produto de perfumaria no estoque.
- **Entrada:** JSON contendo `id`, `nome`, `preco` e `fragrancia`.
- **Saída:** Confirmação de sucesso ou mensagem de erro.

5.3 Listagem de Estoque

- **Rota:** `GET /estoque`

- **Descrição:** Retorna todos os produtos cadastrados no estoque.
- **Saída:** JSON com a lista de produtos.

5.4 Realização de Venda

- **Rota:** `POST /vendas`
- **Descrição:** Realiza uma venda associando um cliente a uma lista de produtos existentes.
- **Entrada:** JSON contendo `cpf`, `nome` e lista de `pids` (IDs dos produtos).
- **Saída:** Dados da venda realizada.

---

6. Cliente da Aplicação

O cliente foi implementado em JavaScript utilizando Node.js e a biblioteca Axios. Ele realiza chamadas HTTP para todos os serviços da API, demonstrando a comunicação cliente-servidor exigida pelo trabalho.

As operações executadas pelo cliente incluem:

- Cadastro de medicamento
- Cadastro de perfumaria
- Consulta ao estoque
- Realização de venda

Os resultados retornados pela API são exibidos no console do cliente.

---

7. Execução do Sistema

1. O servidor Flask é iniciado na porta 5000.
2. O cliente Node.js realiza requisições HTTP para a API.
3. A API processa as requisições e retorna respostas em formato JSON.
4. O cliente exibe os resultados no terminal.

---

8. Conclusão

Com a implementação deste trabalho, foi possível aplicar conceitos fundamentais de sistemas distribuídos, como comunicação cliente-servidor, Web Services, APIs REST e serialização de dados em JSON. A utilização de diferentes linguagens de programação para cliente e servidor reforça a interoperabilidade proporcionada por APIs baseadas em HTTP.

O sistema atende a todos os requisitos propostos, funcionando corretamente e demonstrando, de forma prática, a implementação de um serviço distribuído.