Lista de compras:
Remote Procedure Call (RPC)

Jênifer Vieira Goedert[1]

[1]Acadêmica do Curso de Bacharelado em Ciência da Computação
do Instituto Federal Catarinense - Campus Rio do Sul, SC - Brasil

jenifergoedert10@gmail.com

Abstract. *This article addresses the development of a shopping list application, which meets the established requirements through the use of Remote Procedure Call (RPC). The application provides users with a basic system to add items to the shopping list, view them, and perform calculations based on the registered items, such as the total value and total item count. The project was conceived and implemented using the Python programming language, with data storage in JSON format. Throughout this work, it is possible to comprehensively understand the structure of an application based on Remote Procedure Call (RPC), which promotes agility and efficiency in communication in Distributed Systems.*

Resumo. *Este artigo aborda o desenvolvimento de uma aplicação de lista de compras, que atende aos requisitos estabelecidos através do uso de Remote Procedure Call (RPC). A aplicação proporciona ao usuário um sistema básico para adicionar itens à lista de compras, visualizá-los e realizar cálculos com base nos itens cadastrados, como a soma do valor total e a contagem total de itens. O projeto foi concebido e implementado utilizando a linguagem de programação Python, com armazenamento de dados em arquivos no formato JSON. Ao longo deste trabalho, é possível compreender de forma abrangente a estrutura de uma aplicação baseada em Chamada de Procedimento Remoto (RPC), que promove agilidade e eficiência na comunicação em Sistemas Distribuídos.*

1. Introdução

A Chamada de Procedimento Remoto (RPC) é uma tecnologia para a criação de programas distribuídos servidor/cliente que provê um paradigma de comunicação de alto nível no sistema operacional, já presumindo a existência de um protocolo de transporte, como TCP/IP ou UDP, para carregar a mensagem entre os programas comunicantes. Desta forma, o usuário pode utilizar procedimentos remotos como se fossem chamadas locais, sem precisar se preocupar com possíveis diferenças, tal como a arquitetura, da máquina do processo cliente e do servidor [UEP 2024].

Neste trabalho, a abordagem RPC selecionada foi RESTful. Uma API RESTful é uma maneira segura e eficiente para sistemas de computador trocarem informações pela internet. Ela define regras para comunicação entre sistemas, permitindo que diferentes aplicativos compartilhem recursos de forma confiável. Uma API, ou interface de programação de aplicativos, especifica como os sistemas podem interagir entre si, enquanto REST, ou Representational State Transfer, é uma arquitetura de software que estabelece diretrizes para essa interação. As APIs RESTful seguem os princípios do REST,

incluindo uma interface uniforme, ausência de estado, sistema em camadas, capacidade de armazenamento em cache e a possibilidade de enviar código sob demanda para os clientes. Esses princípios garantem uma comunicação consistente, escalável e eficiente entre os sistemas [AMA 2023].

Essa abordagem foi escolhida devido à sua versatilidade, já que APIs REST podem ser criadas utilizando praticamente qualquer linguagem de programação e suportam diversos formatos de dados. Isto, aliado à comodidade e simplicidade de desenvolvimento, torna-o uma escolha favorável. Além disso, há uma comunidade altamente ativa. Desta forma, as APIs RESTful oferecem benefícios como escalabilidade, flexibilidade e independência tecnológica. Elas facilitam a escalabilidade dos sistemas ao otimizar a comunicação entre cliente e servidor, garantindo uma performance consistente mesmo em grandes volumes de dados. Além disso, permitem a separação total entre cliente e servidor, possibilitando a evolução independente de cada componente sem afetar a outra parte. Isso também as torna independentes da tecnologia utilizada, permitindo que diferentes linguagens de programação sejam usadas tanto no cliente quanto no servidor, sem comprometer a integração[2]. Entretanto, as desvantagens das APIs REST incluem aumento da complexidade do projeto, exigência de conexão com a Internet para qualquer modificação e desempenho variável devido à dependência dos servidores e velocidade da rede. Além disso, sua arquitetura pode limitar a flexibilidade em comparação com outras APIs [THE 2024].

2. Diagramação

O diagrama UML foi elaborado contemplando as entidades principais do projeto: Pessoa, Lista e Item. A Pessoa é identificada por um ID único e possui um nome, além de estar associada a uma Lista específica. A aplicação é projetada para suportar apenas uma Pessoa, que, por sua vez, pode ter apenas uma Lista associada. A entidade Lista é caracterizada por um ID único, um conjunto de Itens, data de criação, valor total e quantidade total de itens. Os métodos de inserção e listagem de itens são implementados nesta entidade. Destaca-se que cada Item está vinculado exclusivamente a uma Lista, conforme a escolha de projeto. Por fim, a entidade Item é composta por uma descrição, quantidade e preço. A utilização de UUID para gerar os IDs garante a unicidade dos identificadores no sistema, proporcionando uma forma robusta de identificação de objetos ou entidades. Essa abordagem promove a coesão e integridade do sistema, assegurando a singularidade de cada entidade. Veja o diagrama UML a seguir.

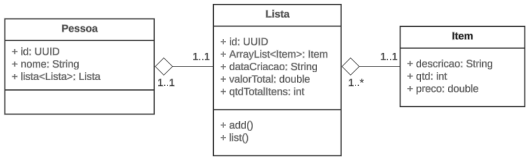

Figure 1. Diagrama UML

O projeto foi organizado em três camadas distintas: Cliente, Negócio e Dados. Na camada do Cliente, ocorrem as interações com a interface e as requisições para as APIs de negócio e dados. Na camada de Negócio, encontra-se a API ”negocioAPI” rodando na porta 8001, as entidades, cálculos como a quantidade total de itens e a soma do valor

total da compra, bem como o gerador de UUID. Por fim, a camada de Dados contém a API ”dadosAPI” rodando na porta 8000 e é responsável pelo gerenciamento dos dados em formato JSON, incluindo a criação de pessoas, itens e a gravação dos dados.

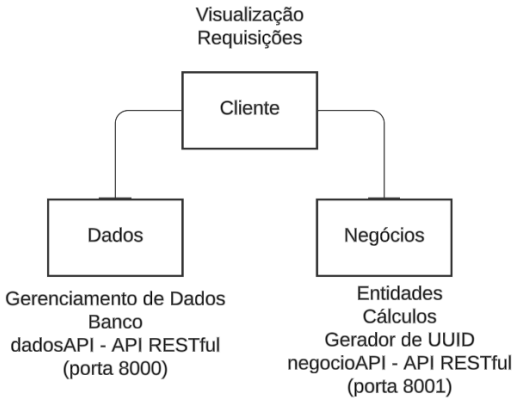

Figure 2. Diagrama de Componentes

3. Desenvolvimento em Python

De forma geral, a aplicação é composta por duas partes principais: uma API de dados (dadosAPI) e uma API de negócios (negocioAPI). A API de dados gerencia operações de dados, como criação de pessoas e itens, enquanto a API de negócios realiza cálculos e operações de negócios, como calcular o total de itens e o valor total da lista de compras de uma pessoa. Todos os dados são gerenciados de forma assíncrona, utilizando uma biblioteca httpx para realizar requisições HTTP assíncronas. Isso permite que o sistema faça requisições HTTP de forma não bloqueante, melhorando a capacidade de resposta e a eficiência geral do sistema. Além disso, uma programação assíncrona pode levar a um uso mais eficiente dos recursos do sistema, pois permite que o sistema execute outras tarefas enquanto espera por operações de E/S para serem concluídas, em vez de ficar ocioso.

Dentre o código, o trecho selecionado foi a adição de itens na lista. A adição de itens é realizada através de uma requisição HTTP POST assíncrona para o endpoint ”/postItem/”. Esta requisição é feita utilizando uma biblioteca httpx. A requisição inclui um payload JSON contendo os detalhes da pessoa e os itens a serem adicionados à sua lista de compras. O payload é estruturado de acordo com os modelos definidos no sistema (PessoaModel), garantindo que os dados enviados sejam corretamente formatados e validados. A resposta da requisição é um objeto JSON que representa uma pessoa atualizada, incluindo a lista de compras com os novos itens adicionados. Esta resposta é gerada pelo endpoint ”/postItem/” da API de dados, que processa a requisição, adiciona os itens à lista de compras da pessoa e retorna a pessoa atualizada, de acordo com o modelo estruturado. Veja na figuras 3 e 4 o código fonte da requisição POST enviado à API de dados e sua rota.

No entanto, para a API de negócios, a requisição para a realização dos cálculos é feita passando como parâmetro uma estrutura de Pessoa. O endpoint ”/calculos/” é responsável por processar essas informações, retornando a Pessoa com a lista de compras atualizada, incluindo os valores de soma total da compra e quantidade total de itens.

Posteriormente, uma requisição é feita à API de dados para a gravação das informações atualizadas.

```python
itens = []

while True:
    descricao = input("Digite a descrição: ")
    qtd = int(input("Digite a quantidade: "))
    preco = float(input("Digite o preço: "))

    item = {
        "id": str(gerador.gerarUUID()),
        "descricao": descricao,
        "qtd": qtd,
        "preco": preco
    }
    itens.append(item)

    continuar = input("1 - Continuar\n2 - Parar\nDigite sua escolha: ")

    if continuar == '2':
        break

async def postItem(pessoa, itens):
    url = "http://localhost:8000/postItem/"
    payload = {
        "pessoa": {
            "id": pessoa['id'],
            "nome": {"nome": pessoa['nome']['nome']},
            "lista": {
                "id": pessoa['lista']['id'],
                "listaItens": [],
                "dataCriacao": pessoa['lista']['dataCriacao'],
                "valorTotal": pessoa['lista']['valorTotal'],
                "qtdTotalItens": pessoa['lista']['qtdTotalItens']
            }
        },
        "itens": itens
    }
    async with httpx.AsyncClient() as client:
        response = await client.post(url, json=payload)
        return response.json()

pessoa = asyncio.run(postItem(pessoa, itens))
```

Figure 3. Código fonte - Requisição POST para adicionar item

```python
@dadosAPI.post("/postPessoa/")
def addPessoa(nome: PessoaInput):
      pessoa = MovimentacaoDados.addPessoa(nome)
      lista = MovimentacaoDados.criaLista()
      pessoa.setLista(lista)
      return pessoa
```

Figure 4. Código fonte - Endpoint"/postItem/"

4. Conclusão

Após a análise e desenvolvimento da aplicação de lista de compras utilizando a abordagem de Chamada de Procedimento Remoto (RPC), demonstrou-se de certa forma, que o desenvolvimento do sistema pode ser considerado relativamente simples. Mas que com maior complexidade do projeto, a implementação se torna mais complexa também. No entanto, a estruturação em camadas e a utilização de APIs RESTful são fundamentais para o desenvolvimento de sistemas distribuídos, promovendo modularidade, escalabilidade e flexibilidade. A transparência na transição de dados entre locais diferentes é crucial, mantendo a experiência do usuário coesa, independentemente da localização física dos componentes do sistema. Essa abordagem permite que sistemas distribuídos sejam específicos para lidar eficazmente com desafios como escalabilidade, tolerância a falhas e segurança, garantindo a eficácia, eficiência e agilidade na comunicação e no processamento de dados.

References

(2023). O que é uma api restful? Acesso em: 08 de abr. de 2024.

(2024). Remote procedure call. Acesso em: 08 de abr. de 2024.

(2024). Rest api: what it is, how it works, advantages and disadvantages. Acesso em: 09 de abr. de 2024.