LabRPC: Laboratório de Remote Procedure Call (RPC)

João Vitor Lehmen Sanmartin[1]

[1]Acadêmico do Curso de Bacharel em Ciência da Computação
Instituto Federal Catarinense Campus Rio do Sul

jvsanmartin@yahoo.com.br

Abstract. *This article presents LabRPC, a Remote Procedure Call (RPC) laboratory using the RESTful pattern. LabRPC is a web application that utilizes FastAPI to create a web API interface, with RPC-based communication for remote interactions between clients and servers. The article explores the LabRPC architecture, including UML and component diagrams, as well as comments on parts of the source code.*

Resumo. *Este artigo apresenta o LabRPC, um laboratório de Remote Procedure Call (RPC) utilizando o padrão RESTful. O LabRPC é uma aplicação web que utiliza o FastAPI para criar uma interface de API web, com comunicação baseada em RPC para interações remotas entre clientes e servidores. O artigo explora a arquitetura do LabRPC, incluindo diagramas UML e de componentes, além de comentários sobre partes do código-fonte.*

1. Introdução

Neste artigo, exploramos o LabRPC, um laboratório de Remote Procedure Call (RPC) focado em comunicação RESTful. O LabRPC utiliza o FastAPI para criar uma interface de API web, permitindo interações remotas entre clientes e servidores de forma eficiente e organizada. Além disso, são apresentados diagramas UML e de componentes para ilustrar a arquitetura do LabRPC.

2. Comentando o Código Fonte

Um trecho de código importante em nosso projeto é a definição das classes `Produto`, `Categoria` e `Fabricante` no arquivo `business.py`. Essas classes representam entidades principais do sistema, como produtos, categorias e fabricantes.

```
1 import uuid
2
3 class Produto:
4     def __init__(self, id=None, nome=None, descricao=None,
          preco=None, quantidade=None, fabricante_id=None,
          categoria_id=None):
5         self.id = id if id else uuid.uuid4()
6         self.nome = nome
7         self.descricao = descricao
8         self.preco = preco
9         self.quantidade = quantidade
```

```
        self.fabricante_id = fabricante_id
        self.categoria_id = categoria_id

class Categoria:
    def __init__(self, id=None, nome=None):
        self.id = id if id else uuid.uuid4()
        self.nome = nome

class Fabricante:
    def __init__(self, id=None, nome=None):
        self.id = id if id else uuid.uuid4()
        self.nome = nome
```

Outra parte relevante é o arquivo `client.py`, que define funções para interagir com o servidor utilizando a biblioteca httpx. Isso permite realizar operações como cadastrar produtos, categorias e fabricantes remotamente.

```
import httpx
import asyncio

async def post_produto(nome):
    url = "http://localhost:8000/postProduto/"
    data = {"nome": nome}
    async with httpx.AsyncClient() as client:
        response = await client.post(url, json=data)
        print(response.json())
        return response.json()

async def post_categoria(nome):
    url = "http://localhost:8000/postCategoria/"
    data = {"nome": nome}
    async with httpx.AsyncClient() as client:
        response = await client.post(url, json=data)
        print(response.json())
        return response.json()

async def post_fabricante(nome):
    url = "http://localhost:8000/postFabricante/"
    data = {"nome": nome}
    async with httpx.AsyncClient() as client:
        response = await client.post(url, json=data)
        print(response.json())
        return response.json()
```

3. Diagrama UML

Os diagramas UML e de componentes são essenciais para entender a arquitetura do LabRPC. Eles representam a estrutura das classes e componentes do sistema, mostrando

as relações entre eles e como as informações são organizadas e processadas. O diagrama UML representa as classes essenciais do projeto LabRPC. A classe Produto possui atributos como ID, nome, descrição, preço, quantidade, além de IDs para fabricante e categoria. A classe Categoria tem atributos de ID e nome, enquanto a classe Fabricante possui ID e nome. As relações indicadas no diagrama mostram que um produto pode pertencer a uma categoria e ser fabricado por um fabricante.

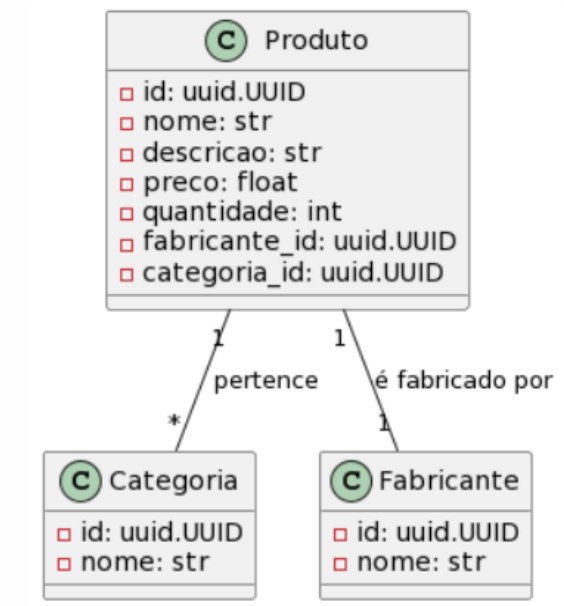

Figure 1. Diagrama UML do LabRPC

4. Diagrama de Componentes

O diagrama de componentes representa a arquitetura do projeto LabRPC, dividido em diferentes pacotes. O pacote ”business” contém as classes Produto, Categoria e Fabricante, representando entidades principais do sistema. O pacote ”client” possui o arquivo client.py, responsável por interagir com o servidor utilizando a biblioteca httpx. O pacote ”data” contém os arquivos main.py e models.py, responsáveis pela manipulação dos dados e integração com o Pydantic para validação dos modelos de dados. O pacote ”LabRPC” utiliza o FastAPI para criar a interface da API web, que se comunica com o httpx para requisições HTTP e com o Pydantic para validação dos dados de entrada e saída. O arquivo JSON é utilizado para armazenar os dados do sistema de forma persistente. As setas indicam as dependências entre os componentes, mostrando a comunicação e interação entre eles no sistema LabRPC.

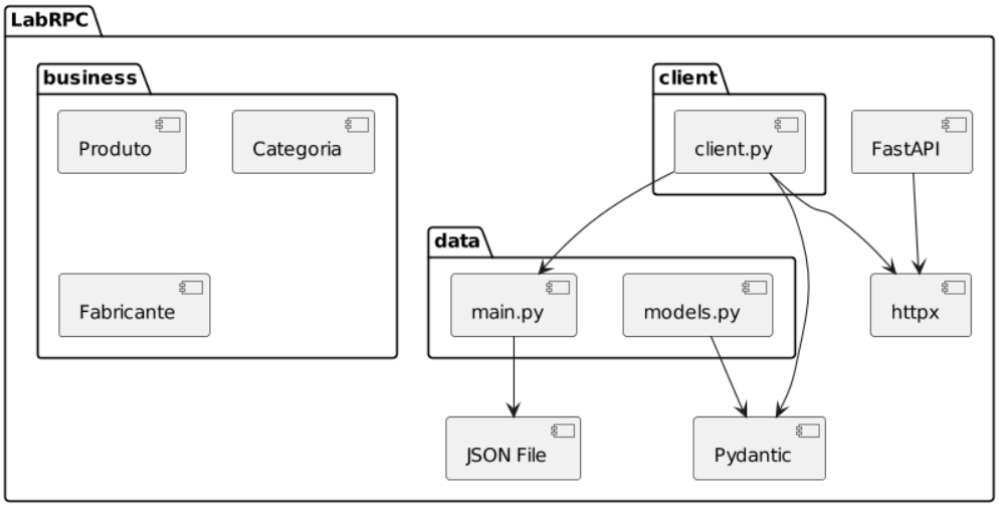

Figure 2. Diagrama de Componentes do LabRPC

5. Conclusão

O LabRPC é um laboratório de Remote Procedure Call (RPC) que utiliza o padrão RESTful e o FastAPI para criar uma interface de API web. Sua arquitetura é baseada em diagramas UML e de componentes, proporcionando uma visão clara da estrutura do sistema e facilitando o desenvolvimento e manutenção do projeto.