Projeto de Gerenciamento de Estoque com Estrutura Cliente-Servidor

Professor: Pedro Capelari

December 20, 2024

Introdução

Este documento descreve o desenvolvimento de um projeto de gerenciamento de estoque usando uma estrutura cliente-servidor. A aplicação é construída em **Python** e integra as seguintes tecnologias:

- **Flask** para criar o servidor e expor uma API.
- **MongoDB** para armazenar os dados do estoque.
- **Tkinter** para a interface gráfica do cliente.
- **Requests** para a comunicação HTTP entre cliente e servidor.

—

1 Estrutura do Projeto

A estrutura de diretórios do projeto é a seguinte:

```
gerenciamento_estoque/
 server/
    server.py
 client/
    client.py
 requirements.txt
```

1.1 Explicação dos Arquivos

- **server/server.py**: Servidor Flask que se conecta ao MongoDB e expõe endpoints para operações de estoque.
- **client/client.py**: Cliente Tkinter que faz requisições HTTP ao servidor.
- **requirements.txt**: Lista das dependências necessárias para o projeto.

—

2 Código do Servidor Flask

```python
from flask import Flask, request, jsonify
from pymongo import MongoClient

app = Flask(__name__)

# Conex o com o MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["estoque_db"]
collection = db["estoque"]

# Rota para adicionar um item
@app.route("/adicionar", methods=["POST"])
def adicionar_item():
    data = request.json
    nome = data["nome"]
    quantidade = data["quantidade"]

    item = collection.find_one({"nome": nome})
    if item:
        nova_quantidade = item["quantidade"] + quantidade
        collection.update_one({"nome": nome}, {"$set": {"quantidade": nova_quantidade}})
    else:
        collection.insert_one({"nome": nome, "quantidade": quantidade})
    return jsonify({"message": f"Item '{nome}' adicionado com sucesso."})

# Rota para remover um item
@app.route("/remover", methods=["POST"])
def remover_item():
    data = request.json
    nome = data["nome"]
    quantidade = data["quantidade"]

    item = collection.find_one({"nome": nome})
    if item:
        nova_quantidade = item["quantidade"] - quantidade
        if nova_quantidade <= 0:
            collection.delete_one({"nome": nome})
        else:
            collection.update_one({"nome": nome}, {"$set": {"quantidade": nova_quantidade}})
        return jsonify({"message": f"Item '{nome}' removido com sucesso."})
    else:
        return jsonify({"error": f"Item '{nome}' n o encontrado."}), 404

# Rota para listar todos os itens
@app.route("/listar", methods=["GET"])
def listar_itens():
    itens = list(collection.find({}, {"_id": 0}))
    return jsonify(itens)

if __name__ == "__main__":
```

```
    app.run(host="0.0.0.0", port=5000, debug=True)
```

Listing 1: Servidor Flask com MongoDB

---

3 Código do Cliente Tkinter

```
import tkinter as tk
from tkinter import ttk, messagebox
import requests

SERVER_URL = "http://localhost:5000"

class EstoqueApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Gerenciamento de Estoque - Cliente")

        # Widgets
        self.nome_label = ttk.Label(root, text="Nome do Item:")
        self.nome_label.grid(row=0, column=0, padx=5, pady=5)
        self.nome_entry = ttk.Entry(root)
        self.nome_entry.grid(row=0, column=1, padx=5, pady=5)

        self.qtd_label = ttk.Label(root, text="Quantidade:")
        self.qtd_label.grid(row=1, column=0, padx=5, pady=5)
        self.qtd_entry = ttk.Entry(root)
        self.qtd_entry.grid(row=1, column=1, padx=5, pady=5)

        self.adicionar_btn = ttk.Button(root, text="Adicionar Item",
    command=self.adicionar_item)
        self.adicionar_btn.grid(row=2, column=0, padx=5, pady=5)

        self.remover_btn = ttk.Button(root, text="Remover Item",
    command=self.remover_item)
        self.remover_btn.grid(row=2, column=1, padx=5, pady=5)

        self.listar_btn = ttk.Button(root, text="Listar Itens", command
    =self.listar_itens)
        self.listar_btn.grid(row=3, column=0, columnspan=2, padx=5,
    pady=5)

        self.resultado_text = tk.Text(root, height=10, width=40)
        self.resultado_text.grid(row=4, column=0, columnspan=2, padx=5,
     pady=5)

    def adicionar_item(self):
        nome = self.nome_entry.get()
        quantidade = int(self.qtd_entry.get())
        response = requests.post(f"{SERVER_URL}/adicionar", json={"nome
    ": nome, "quantidade": quantidade})
        messagebox.showinfo("Resposta", response.json()["message"])

    def remover_item(self):
        nome = self.nome_entry.get()
        quantidade = int(self.qtd_entry.get())
```

```
        response = requests.post(f"{SERVER_URL}/remover", json={"nome":
    nome, "quantidade": quantidade})
        if response.status_code == 200:
            messagebox.showinfo("Resposta", response.json()["message"])
        else:
            messagebox.showerror("Erro", response.json()["error"])

    def listar_itens(self):
        response = requests.get(f"{SERVER_URL}/listar")
        self.resultado_text.delete(1.0, tk.END)
        itens = response.json()
        for item in itens:
            self.resultado_text.insert(tk.END, f"{item['nome']}: {item
    ['quantidade']}\n")

if __name__ == "__main__":
    root = tk.Tk()
    app = EstoqueApp(root)
    root.mainloop()
```

Listing 2: Cliente Tkinter com Requisições HTTP

—

4 Instalação e Execução

4.1 Instalação das Dependências

Crie um arquivo `requirements.txt` com o seguinte conteúdo:

```
Flask==3.0.0
pymongo==4.6.1
requests==2.31.0
```

Instale as dependências com:

```
pip install -r requirements.txt
```

4.2 Iniciar o MongoDB

```
sudo service mongod start
```

4.3 Iniciar o Servidor Flask

```
cd server
python server.py
```

4.4 Executar o Cliente Tkinter

```
cd client
python client.py
```

—

5 Conclusão

Este projeto demonstra como construir uma aplicação de gerenciamento de estoque usando uma estrutura cliente-servidor com Flask, MongoDB e Tkinter. Ele segue boas práticas de desenvolvimento de software, promovendo a separação de responsabilidades e a escalabilidade.