re_S9_A2_SL06_backend

Roteiro de Atividade Prática

Nome: _____ Turma: _____

Título da atividade: GraphQL

Objetivo:
Desenvolver APIs utilizando GraphQL, explorando os conceitos de consultas (queries), mutações e a flexibilidade oferecida por GraphQL em comparação com APIs RESTful, possibilitando a recuperação precisa de dados conforme a necessidade do cliente.

Conceito técnico: GraphQL

GraphQL é uma linguagem de consulta para APIs que viabiliza ao cliente especificar exatamente quais dados deseja recuperar do servidor, ao contrário das APIs RESTful, em que o servidor define os dados retornados em cada endpoint. Em GraphQL, existem três componentes principais:

- Queries: usadas para solicitar dados;
- Mutations: usadas para modificar dados;
- Schemas: definem a estrutura dos dados disponíveis na API, estabelecendo os tipos e as relações entre os dados.

Uma das principais vantagens de GraphQL é a capacidade de evitar over-fetching (quando o cliente recebe mais dados do que o necessário) e under-fetching (quando o cliente precisa fazer múltiplas requisições para obter todos os dados necessários).

Objetivo da atividade:
O participante vai implementar uma API GraphQL para gerenciar produtos de um sistema de e-commerce. A API deve possibilitar ao cliente consultar produtos específicos, bem como ao usuário adicionar novos produtos e atualizar ou remover produtos existentes.

Enunciado:
Você foi encarregado de criar uma API GraphQL para gerenciar os produtos de um sistema de e-commerce. A API deve possibilitar as seguintes operações:

- **Query** – Recuperar todos os produtos ou um produto específico por ID;
- **Mutation** – Adicionar, atualizar e remover produtos.

Implemente essas funcionalidades utilizando **Python** com **Graphene** (biblioteca GraphQL para Python), garantindo que a API siga as boas práticas de GraphQL.

Código de programação:
Abaixo está um exemplo de como implementar essa API GraphQL utilizando **Python** com a biblioteca **Graphene**.

Exemplo em Python com Graphene:

```
import graphene

# Modelo de produto
class Produto(graphene.ObjectType):
    id = graphene.Int()
    nome = graphene.String()
    preco = graphene.Float()

# Lista inicial de produtos (simulando um banco de dados)
produtos = [
    {"id": 1, "nome": "Camiseta", "preco": 50.00},
    {"id": 2, "nome": "Tênis", "preco": 120.00}
]

# Queries (Consultas)
class Query(graphene.ObjectType):
    produto = graphene.Field(Produto, id=graphene.Int())
    produtos = graphene.List(Produto)

    # Retorna um produto por ID
```

```
    def resolve_produto(root, info, id):
        return next((p for p in produtos if p['id'] == id), None)

    # Retorna todos os produtos
    def resolve_produtos(root, info):
        return produtos

# Mutations (Modificações)
class AdicionarProduto(graphene.Mutation):
    class Arguments:
        nome = graphene.String(required=True)
        preco = graphene.Float(required=True)

    produto = graphene.Field(Produto)

    # Adiciona um novo produto
    def mutate(root, info, nome, preco):
        novo_produto = {"id": len(produtos) + 1, "nome": nome, "preco": preco}
        produtos.append(novo_produto)
        return AdicionarProduto(produto=novo_produto)

class Mutation(graphene.ObjectType):
    adicionar_produto = AdicionarProduto.Field()

# Schema
schema = graphene.Schema(query=Query, mutation=Mutation)

# Servidor Flask para rodar a API GraphQL
from flask import Flask, request
from flask_graphql import GraphQLView

app = Flask(__name__)
app.add_url_rule('/graphql', view_func=GraphQLView.as_view('graphql',
schema=schema, graphiql=True))

if __name__ == '__main__':
```

```
app.run(debug=True)
```

Explicação técnica:

- Queries – O código define uma consulta para recuperar um único produto por ID (produto) e outra para listar todos os produtos (produtos). GraphQL viabiliza que o cliente especifique quais campos deseja retornar, otimizando o uso de dados;
- Mutations – A mutação AdicionarProduto garante a adição de novos produtos, recebendo os dados necessários e criando um novo recurso na lista de produtos;
- Schema – O schema combina queries e mutations, definindo a estrutura geral da API. O uso de Graphene facilita a implementação de APIs GraphQL em Python.

Perguntas para conclusão da atividade:

- Explique como o uso de queries no GraphQL pode evitar o problema de over-fetching e under-fetching observado em APIs RESTful.
- Qual é o papel das mutações (mutations) em GraphQL e como elas se diferenciam das queries? Dê um exemplo de uma mutação implementada.
- Descreva o papel do schema em uma API GraphQL e explique como ele ajuda a definir a estrutura da API.