Desafio Python - Desenvolvedor Sênior FIDC

Tempo estimado: 2-3 horas

Contexto

Você precisa construir uma API para processar operações de FIDC; Para tal, é necessário que sua API faça a consulta de preços de ativos e calcule valores das operações.

Sua Missão

Implemente uma API Flask que processe operações financeiras com os seguintes requisitos:

1. API REST (Flask)

Crie um endpoint `POST /operations/process` que:

- Receba uma lista de operações no formato JSON
- Processe cada operação de forma assíncrona
- Retorne um job_id para acompanhar o progresso

Payload de entrada:

```json
{
  "fidc_id": "FIDC001",
  "operations": [
    {
      "id": "op_001",
      "asset_code": "PETR4",
```

```
      "operation_type": "BUY",
      "quantity": 1000,
      "operation_date": "2024-09-01"
    }
  ]
}
```

2. Processamento Assíncrono (Celery)

- Configure Celery com Redis como broker
- Processe operações em background
- Implemente retry para falhas de API externa
- Armazene resultados no banco

3. Integração Externa

Crie uma API simulada para aquisição de preços de ativos (via asset_code) que:

- Falhe 30% das vezes (para testar retry)
- Tenha rate limit de 10 requests/minuto
- Retorne preços aleatórios entre R$10-100

4. Cálculo das Operações

Para cada operação, implemente o seguinte cálculo abaixo (em pseudocódigo):

OBS: Note que o código abaixo não é adequado para uso produtivo, e tem caráter ilustrativo de como o cálculo da operação é feito;

Compra (BUY):

```python
# Buscar preço do ativo na API externa
asset_price
```

```
# Calcular valor bruto
gross_value = quantity * asset_price

# Aplicar taxa fixa (0.5% para compras)
tax_amount = gross_value * 0.005
total_cost = gross_value + tax_amount

# Atualizar caixa do FIDC
fidc.available_cash -= total_cost

# Salvar operação
operation.execution_price = asset_price
operation.total_value = total_cost
operation.tax_paid = tax_amount
```

Venda (SELL):

Python

```
# Buscar preço do ativo
asset_price

# Calcular valor bruto
gross_value = quantity * asset_price

# Aplicar taxa (0.3% para vendas)
tax_amount = gross_value * 0.003
net_proceeds = gross_value - tax_amount

# Atualizar caixa do FIDC
fidc.available_cash += net_proceeds

# Salvar operação
```

```
operation.execution_price = asset_price
operation.total_value = net_proceeds
operation.tax_paid = tax_amount
```

Validações obrigatórias:

- Verificar se FIDC tem caixa suficiente para compras
- Verificar se preço retornado é > 0
- Operações devem ser atômicas (falha = rollback completo)
- Log detalhado para auditoria
- Independente da ordem, o valor do fundo deve ser sempre consistente

5. Banco de Dados (SQLAlchemy)

Modele as tabelas:

- `operations`: id, asset_code, operation_type, quantity, status, execution_price, total_value, tax_paid, created_at
- `processing_jobs`: job_id, status, created_at, completed_at
- `fidc_cash`: fidc_id, available_cash, updated_at

6. Monitoramento

Endpoint `GET /jobs/{job_id}/status` que retorna:

```json
{
  "job_id": "uuid",
  "status": "PROCESSING|COMPLETED|FAILED",
  "total_operations": 100,
  "processed": 75,
  "failed": 2,
  "estimated_completion": "2024-09-01T15:30:00Z"
}
```

7. Exportação em batch para bucket

Implemente um endpoint `POST /operations/export` que recebe o `fidc_id` e um intervalo de datas, e exporta o transacional realizado para um bucket da S3. Para testes locais, utilize o minio.

Requisitos Técnicos

- Flask + SQLAlchemy + Celery
- Validação de dados (Marshmallow/Pydantic)
- Minio / S3
- Logging estruturado
- Tratamento de exceções
- Testes unitários básicos
- Docker Compose para subir ambiente completo

Entregáveis

1. **Código fonte** completo e funcional
2. **README.md** com instruções para rodar
3. **docker-compose.yml** para ambiente completo
4. **Testes** para pelo menos uma função crítica
5. **Exemplo de uso** da API (curl ou Postman)

Não Precisa Implementar

- Autenticação/autorização
- Interface web
- Deploy em produção
- Métricas avançadas

Avaliação

- **Código limpo e organização** (30%)
- **Funcionamento correto** (25%)
- **Tratamento de erros** (20%)

- **Configuração de ambiente** (15%)
- **Testes** (10%)

Dicas

- Foque na funcionalidade core, não em features extras
- Documente decisões técnicas no README
- Use variáveis de ambiente para configuração
- Organize o código em módulos lógicos

Atenção

❌ Código que não roda
❌ Sem tratamento de erros
❌ Configuração hardcoded
❌ Sem documentação básica
❌ Estrutura confusa de arquivos

Envie o código via GitHub e inclua instruções claras para executar.