**Introdução**

O objetivo deste projeto é desenvolver um sistema distribuído capaz de simular o processo de validação por consenso de uma moeda digital, denominada NoNameCoin. O sistema será modelado utilizando o modelo de consenso Proof of Stake (PoS), onde os nós da rede são selecionados para validar transações com base na quantidade de moeda que possuem e "apostam" como garantia.

**Estrutura do Projeto**

O sistema é composto por três camadas principais:

1. **Banco**: Gerencia contas e transações.
2. **Seletor**: Seleciona validadores e gerencia o consenso.
3. **Validador**: Valida transações de acordo com as regras do sistema.

**Organização do Código**

A estrutura do projeto é organizada em diretórios e arquivos para facilitar a manutenção e escalabilidade:

markdown

Copiar código

no\_name\_coin/ ├── app.py ├── banco/ │ ├── \_\_init\_\_.py │ └── routes.py ├── seletor/ │ ├── \_\_init\_\_.py │ └── routes.py ├── validador/ │ ├── \_\_init\_\_.py │ └── routes.py └── utils/ ├── \_\_init\_\_.py └── helpers.py

**1. Arquivo app.py**

Este é o arquivo principal que inicializa o aplicativo Flask e registra os Blueprints das diferentes camadas.

python

Copiar código

from flask import Flask from banco.routes import banco\_bp from seletor.routes import seletor\_bp from validador.routes import validador\_bp app = Flask(\_\_name\_\_) # Registro dos Blueprints app.register\_blueprint(banco\_bp, url\_prefix='/banco') app.register\_blueprint(seletor\_bp, url\_prefix='/seletor') app.register\_blueprint(validador\_bp, url\_prefix='/validador') if \_\_name\_\_ == '\_\_main\_\_': app.run(debug=True)

**2. Diretório banco/routes.py**

Esta camada gerencia as contas e transações, verificando saldo, tempo e limite de transações.

python

Copiar código

from flask import Blueprint, request, jsonify import datetime banco\_bp = Blueprint('banco', \_\_name\_\_) accounts = { 'user1': {'balance': 1000, 'last\_transaction\_time': None, 'transaction\_count': 0}, 'user2': {'balance': 500, 'last\_transaction\_time': None, 'transaction\_count': 0} } def validate\_transaction(sender, receiver, amount, fee, trans\_time): if sender not in accounts or receiver not in accounts: return 2, 'Conta inválida' sender\_account = accounts[sender] receiver\_account = accounts[receiver] if sender\_account['balance'] < amount + fee: return 2, 'Saldo insuficiente' if sender\_account['last\_transaction\_time'] and trans\_time <= sender\_account['last\_transaction\_time']: return 2, 'Horário da transação inválido' if sender\_account['transaction\_count'] > 100: return 2, 'Limite de transações excedido' return 1, 'Transação válida' @banco\_bp.route('/') def index(): return "Banco API" @banco\_bp.route('/trans', methods=['POST']) def transaction(): data = request.get\_json() sender = data['sender'] receiver = data['receiver'] amount = data['amount'] fee = data['fee'] trans\_time = datetime.datetime.fromisoformat(data['time']) status, message = validate\_transaction(sender, receiver, amount, fee, trans\_time) if status == 1: accounts[sender]['balance'] -= amount + fee accounts[receiver]['balance'] += amount accounts[sender]['last\_transaction\_time'] = trans\_time accounts[sender]['transaction\_count'] += 1 return jsonify({'status': status, 'message': message}) @banco\_bp.route('/hora') def hora(): current\_time = datetime.datetime.now().isoformat() return jsonify({'current\_time': current\_time})

**3. Diretório seletor/routes.py**

Esta camada seleciona validadores para uma transação e gerencia o consenso.

python

Copiar código

from flask import Blueprint, request, jsonify seletor\_bp = Blueprint('seletor', \_\_name\_\_) validators = { 'validator1': {'stake': 100, 'flag': 0, 'active': True, 'selected\_count': 0}, 'validator2': {'stake': 200, 'flag': 0, 'active': True, 'selected\_count': 0}, 'validator3': {'stake': 150, 'flag': 0, 'active': True, 'selected\_count': 0} } @seletor\_bp.route('/cadastrar', methods=['POST']) def cadastrar\_validador(): data = request.get\_json() validator\_id = data['validator\_id'] stake = data['stake'] if validator\_id in validators: return jsonify({'status': 2, 'message': 'Validador já cadastrado'}) validators[validator\_id] = {'stake': stake, 'flag': 0, 'active': True, 'selected\_count': 0} return jsonify({'status': 1, 'message': 'Validador cadastrado com sucesso'}) def select\_validators(): active\_validators = {k: v for k, v in validators.items() if v['active']} sorted\_validators = sorted(active\_validators.items(), key=lambda item: item[1]['stake'], reverse=True) selected = sorted\_validators[:3] for validator\_id, \_ in selected: validators[validator\_id]['selected\_count'] += 1 if validators[validator\_id]['selected\_count'] >= 5: validators[validator\_id]['active'] = False return selected @seletor\_bp.route('/eleger', methods=['POST']) def eleger\_validadores(): selected\_validators = select\_validators() validator\_ids = [vid for vid, \_ in selected\_validators] return jsonify({'status': 1, 'validators': validator\_ids})

**4. Diretório validador/routes.py**

Esta camada valida transações de acordo com as regras do sistema.

python

Copiar código

from flask import Blueprint, request, jsonify validador\_bp = Blueprint('validador', \_\_name\_\_) validators = { 'validator1': {'stake': 100, 'flag': 0, 'active': True, 'selected\_count': 0}, 'validator2': {'stake': 200, 'flag': 0, 'active': True, 'selected\_count': 0}, 'validator3': {'stake': 150, 'flag': 0, 'active': True, 'selected\_count': 0} } def validate\_transaction\_by\_validator(transaction, validator\_id): validator = validators.get(validator\_id) if validator and validator['active']: return 1, 'Transação validada pelo validador' return 2, 'Validador inativo ou não encontrado' @validador\_bp.route('/validar', methods=['POST']) def validar\_transacao(): data = request.get\_json() transaction = data['transaction'] validator\_id = data['validator\_id'] status, message = validate\_transaction\_by\_validator(transaction, validator\_id) return jsonify({'status': status, 'message': message})

**Configuração do Ambiente**

**Instalação de Dependências**

Instale o Flask e outras bibliotecas necessárias:

bash

Copiar código

pip install flask sqlalchemy flask-jwt-extended flasgger

**Configuração do Banco de Dados**

Para persistência dos dados, você pode usar um banco de dados como SQLite, PostgreSQL ou qualquer outro de sua preferência. Use o SQLAlchemy para interagir com o banco de dados.

**Autenticação e Segurança**

Implemente autenticação e autorização para proteger suas rotas. Use tokens JWT para autenticação segura.

bash

Copiar código

pip install flask-jwt-extended

**Documentação da API**

Use ferramentas como Swagger ou Postman para documentar e testar suas APIs.

bash

Copiar código

pip install flasgger

**Recursos Adicionais**

**Persistência de Dados**

Configure um banco de dados para armazenar informações de contas, transações e validadores de forma persistente. Utilize SQLAlchemy para interagir com o banco de dados.

**Autenticação**

Implemente autenticação de usuários e validadores para proteger suas rotas. Utilize JWT para autenticação segura.

**Testes**

Escreva testes unitários e de integração para garantir que suas rotas e funcionalidades funcionem corretamente. Utilize bibliotecas como **unittest** ou **pytest**.

**Documentação**

Documente sua API usando Swagger para facilitar o entendimento e a utilização por outros desenvolvedores. Configure o Flask-Swagger para gerar automaticamente a documentação da API.

**Deployment**

Configure o ambiente de produção para hospedar seu serviço web. Utilize contêineres Docker para empacotar e distribuir sua aplicação de forma consistente.

Com esses componentes e considerações adicionais, você terá uma base sólida para iniciar o desenvolvimento do seu sistema distribuído de validação por consenso para a moeda digital NoNameCoin.