Relatório do problema 3:
Transações Bancárias Distribuídas

Anésio Sousa dos Santos Neto[1]

[1]Departamento de tecnologia
Universidade Estadual de Feira de Santana (UEFS)
Av. Transnordestina, s/n, Novo Horizonte – 44036-900
Feira de Santana – BA – Brasil

anesios98@gmail.com

Abstract. *This report describes all the steps for the development of a distributed transaction system, using the MVC architecture pattern, the Flask framework and the logical clock model provided by vector clocks. The entire project was developed using the git versioning tool and all the API's functionalities and endpoints were tested using the Insomnia software.*

Resumo. *Este relatório descreve todos os passos para o desenvolvimento de um sistema de transações distribuidas, utilizando o padrão de arquitetura MVC, o framework Flask e o modelo de relógio lógico proporcionado pelos relógios vetoriais. Todo o projeto foi desenvolvido utilizando a ferramenta de versionamento git e todas as funcionalidades e endpoits das API's foram testados utilizando o software Insomnia.*

1. Introdução

O Pix, sistema de pagamentos instantâneos desenvolvido pelo Banco Central do Brasil, revolucionou a forma como realizamos transações financeiras. Desde o seu lançamento, o Pix tem ganhado cada vez mais popularidade e se tornou uma alternativa segura, eficiente e conveniente para transferências de dinheiro. Uma das vantagens mais marcantes do Pix é a rapidez e agilidade nas transações. Ao contrário dos métodos tradicionais de transferência, como TEDs e DOCs, que podem levar horas ou até mesmo dias para serem concluídos, o Pix permite a transferência instantânea de valores, 24 horas por dia, 7 dias por semana. O Pix tem trazido diversas vantagens para o cenário financeiro brasileiro. Sua rapidez, facilidade, baixo custo, inclusão financeira e estímulo à economia digital são apenas algumas das muitas vantagens que tornam esse sistema uma escolha popular entre os consumidores e empresas. À vista disso, um governo de um país onde não existe banco cventral está interessado em desenvolver um sistema semelhante ao Pix utilizado no Brasil. Esse relatório descreve a construção desse sistema em todas as fases do processo.

2. Fundamentação teórica

Para criar o sistema foi necessário reunir algumas informações vitais para o cumprimento do que foi proposto, escolhendo modelos de projeto distribuido que condizem com a realidade do problema e ferramentas para o melhor desenvolvimento do produto. Ferramentas como: linguagem a ser utilizada, frameworks, padrões de projeto a serem utilizados, estruturas de dados e algoritmos de ordenação de dados.

2.1. Padrão arquitetural

O padrão arquitetural escolhido para o desenvolvimento do sistema foi o MVC. A divisão em camadas que esse padrão prega, proporciona ao sistema uma melhor distribuição e entendimento do fluxo de entrada e saida. Permitindo que sejam incrementadas novas funcionalidades rapidamente, tornando o sistema mais manutenível e mais simples de ser refatorado.

2.2. Requests for Humans

O ”requests” é uma biblioteca popular do Python para fazer requisições HTTP de forma fácil e flexível. Ela é muito utilizada para interagir com APIs, enviar dados para servidores web, realizar raspagem de dados, entre outras tarefas relacionadas à comunicação com a web.

O termo ”requests for humans” refere-se ao lema do projeto, que destaca o objetivo de criar uma biblioteca simples e amigável para uso humano, facilitando a realização de solicitações HTTP de maneira intuitiva e compreensível.

2.3. Lamport clock

Para criar a idéia de causalidade em sistemas distribuidos torna-se necessário capturar a idéia de um evento acontecer antes de outro. O cientista da computação Leslie Lamport estabeleceu uma forma de realizar esse feito e o chamou de relógio lógico. Atualmente a relógios assim usa-se o nome relógio de Lamport. Os relógios desse tipo permitem associar *timestamps* a eventos de forma que garantam as seguintes propriedades [Lasaro 2021]:

- Seja *e* um evento
- Seja *C(e)* o valor do relógio associado a *e*
- Se $e \rightarrow e'$ então *C(e)* $<$ *C(e')*

Essa função C é definida assim:

- Seja C um contador de processo com valor inicialmente igual a 0 e associado a um processo p.
- *C(e)* = ++*C de p* no momento em que *e* ocorreu
- Podemos usar o símbolo $<$ para comparar eventos, funcionando da mesma forma quando avaliamos dois inteiros.

Fica claro que não há uma fonte da verdade como um relógio na parede. Cada processo mantém seu próprio relógio que pode ser relacionado com relógios de outros processos[Lasaro 2021].

Para alguns tipos de eventos a terceira propriedade não é respeitada. Para que essa propriedade se mantenha válida, precisamos modificar a tentativa anterior de definição anterior para que, na recepção de uma mensagem, os contadores sejam atualizados para que sejam maiores tanto que os relógios dos eventos locais quanto dos eventos que antecederam o envio da mensagem sendo recebida[Lasaro 2021]:

- Seja C um contador de processo associado a um processo p com valor inicialmente igual a 0.
- Se o evento é uma operação local

  - $Cp \leftarrow Cp + 1$
  - *C(e)* $\leftarrow Cp$
- Se o evento *e* é o envio de uma mensagem
  - $Cp \leftarrow Cp + 1$
  - *C(e)* $\leftarrow Cp$
  - *C(e)* é enviado com a mensagem como seu timestamp.
- Se o evento *e* é a recepção de uma mensagem com timestamp *ts*
  - $Cp \leftarrow max(Cp, ts) + 1$
  - *C(e)* $\leftarrow Cp$

Com isso obtemos a definição formal de um relógio de Lamport.

2.4. Vector clock

O relógio de Lamport, porém, é incompleto. Relógios vetoriais são relógios lógicos em que cada processo mantém não apenas um contador dos seus eventos locais, mas também sua visão dos contadores dos outros processos. Estas visões são atualizadas a cada recepção de mensagem, de acordo com a seguinte especificação, onde assume-se que *n* processos fazem parte do sistema[Lasaro 2021]:

- Seja Cp[i], e $1 \leq i \leq n$ inicialmente igual a 0
- Seja um evento *e*
  - Se *e* é uma operação local
    - $Cp[p] \leftarrow Cp[p] + 1$
    - *V(e)* $\leftarrow Cp$
  - Se *e* é o envio de uma mensagem
    - $Cp[p] \leftarrow Cp[p] + 1$
    - *V(e)* $\leftarrow Cp$
    - *V(e)* é enviado com a mensagem como seu timestamp
  - Se *e* é a recepção de uma mensagem com timestamp *ts* de *q*, então
    - $Cp[p] \leftarrow Cp[p] + 1$
    - *Cp[i]* $\leftarrow max(Cp[i], ts[i]), \forall i \neq p$
    - *V(e)* $\leftarrow Cp$

É possível comparar dois vetores com o operador de igualdade e com os relacionais menor que, maior que e combinados com igualdade. Sendo assim, para dois eventos *e* $\neq$ *e'*:

- $e \rightarrow e' \Leftrightarrow V(e) < V(e')$
- Se $V(e) \not< V(e')$ e $V(e') \not< V(e)$, então *e* e *e'* são concorrentes.

2.5. Sincronização

Suponha uma conta que não possui valor algum e que em um determinado momento *t* recebe duas requisições: uma requisição solicita retirada de um valor e outra solicita depósito do mesmo valor. Como não deve ser possível ter em uma conta saldo negativo, o depósito deve ocorrer primeiro e depois a retirada. Como cada evento tem um identificador único (estado de um relógio vetorial) e é possível saber quem veio antes ou depois (comparar numericamente), podemos organizar esses eventos e sincronizar os processos envolvidos para que obtenhamos atomicidade de operações e garantia de funcionamento correto do sistema como um todo.

2.6. Flask

Flask é um framework leve e flexível em Python para desenvolvimento web. Ele permite criar aplicativos web de forma rápida e simples, fornecendo as ferramentas e estruturas básicas necessárias para lidar com rotas, gerenciar solicitações e respostas, além de oferecer suporte a templates e sessões. Algumas características e benefícios do Flask incluem:

- Simplicidade
- Modularidade
- Roteamento: O Flask permite definir rotas para diferentes URLs, direcionando solicitações HTTP para funções específicas.
- Templates:
- Desenvolvimento rápido: Com o Flask, você pode iniciar um novo projeto web rapidamente, sem muita configuração ou complexidade.

Em suma, o Flask é uma opção popular para desenvolvimento web em Python, oferecendo uma abordagem simples e direta para a criação de aplicativos web

2.6.1. Request

O Flask possui um objeto chamado *request* que fornece acesso aos dados enviados pelo cliente durante uma solicitação HTTP. Esse objeto permite acessar parâmetros de URL, dados de formulário, informações de cabeçalho e muito mais.O request também fornece outros métodos e atributos para acessar informações como cookies, métodos HTTP, endereço IP do cliente, entre outros.

2.7. Python

Python é uma linguagem de programação de alto nível, interpretada e de propósito geral. Foi criada por Guido van Rossum e lançada pela primeira vez em 1991. Python se destaca por sua sintaxe simples e legível, o que facilita a leitura e compreensão do código. Python é amplamente adotado em várias áreas e é uma ótima escolha para diferentes projetos devido à sua simplicidade, expressividade e suporte comunitário.

2.8. Insomnia

Insomnia é uma ferramenta de cliente HTTP de código aberto, disponível para Windows, macOS e Linux, que permite testar e depurar APIs de forma fácil e eficiente. É amplamente utilizado por desenvolvedores para realizar solicitações HTTP, visualizar e analisar respostas, e colaborar em projetos de API. A ferramenta oferece uma série de recursos adicionais para facilitar o trabalho com APIs, tornando-a uma escolha popular entre desenvolvedores e equipes de desenvolvimento.

3. Metodologia

Devido a grande variedade de dispositivos e sistemas operacionais diferentes que precisam interagir entre si e de questões como a escalabilidade de sistemas foi escolhido um modelo de comunicação simples, mas extremamente eficaz e escalável onde cada banco(nó), faz requisições para outros nós utilizando a biblioteca requests e quando um nó recebe essa requisição, a trata utilizando o objeto request do framework Flask. Fazendo uso dos relógios vetoriais para a sincronização.[Kurose and Ross 2017]

3.1. MVC

Usando o padrão MVC, O sistema foi subdividido em 4 módulos, sendo eles:

Models: Onde ficam as regras de negócio do sistema. Aqui residem as classes responsáveis por gerir os modelos principais. Nesse módulo está a classe que define o relógio vetorial e suas operações básicas de incremento, comparação com outros relógios e atualização com base em outros relógios.

View (static e template): Onde ficam as classes responsáveis por fazer a interação homem-máquina. Neste módulo ficam os arquivos estáticos (css e js) e também as páginas HTML usadas como template.

Controllers: Nesse módulo reside o controlador principal do sistema chamado de default. Ele é responsável por gerenciar os endpoints de um banco. Quando chega uma requisição em seus endpoints, essa requisição é processada e dependendo das características desse endpoint, podem ocorrer novas requisições para outros endpoits de outros bancos.

4. Resultados e discussão

O resultado do projeto é um sistema distribuido onde cada nó desse sistema representa um banco com contas e que se comunicam para transferir dinheiro tanto entre contas de nós diferentes, quanto entre contas do mesmo nó. É possível criar contas em um banco, realizar depósitos em uma conta, atualizar os dados de uma conta e deletar uma conta em um banco. Na tabela 1 estão listados os principais endpoits de cada banco e sua funcionalidade.

Table 1. Principais endpoints da API

Nome	Propósito	API endpoint	Verbo HTTP
INDEX	Página inicial do sistema	/ ou /index	GET
LOGIN	Página de login do sistema	/login	POST
ACCOUNTS	Retorna todas as contas de um banco	/accounts	GET
TRANSACTIONS	Transações entre contas de um mesmo banco e entre contas de bancos diferentes	/accounts/transaction	POST

Os principais cenários de uso do sistema em relação as tranferências bancárias está listado na próxima sessão do relatório.

4.1. Transferências entre contas

Para transferir dinheiro entre contas de um mesmo banco, basta realizar uma requisição do tipo POST para o endpoint */accounts/transactions* passando no *payload* um arquivo json no seguinte esquema:

```
{
        "from_account_id" : "aqui entra a conta remetente",
```

```
        "to_account_id": "aqui entra a conta destino",
        "value": aqui entra o valor
}
```

Caso ambos id's correspondam a contas no banco em questão, automaticamente é realizada a transferência.

Para transferir dinheiro onde a conta remetente pertence ao banco atual e a conta destinatária pertence a um banco diferente, o esquema é o mesmo, a diferença é que o id do destinatário não vai ser encontrado no banco atual, sendo necessário descobrir em que banco ela pertence, se pertencer a algum.

Finalmente, para transferir dinheiro entre contas de bancos diferentes, o esquema ainda permanece o mesmo. O banco atual não vai encontrar contas com os id's correspondentes e vai realizar o mesmo processo de busca e encontrar a qual banco cada conta pertence e fazer a delegação de transferência.

5. Conclusão

Essa solução é fruto de meses de trabalho de meia dúzia de colaboradores cujo esforço e energia se tornaram vitais para os resultados obtidos. Nesses meses houveram diversas discussões, pesquisas e reuniões presenciais e remotas que foram essenciais para o completo e total entendimento do problema. Uma interface gráfica consumindo as API's e a correta implementação da causalidade entre transações proporcionaria mais robustez e inteligência ao sistem. Agregando ainda mais elegância à solução exposta e se configurariam como próximos passos na continuidade do desenvolvimento do mesmo.

References

Kurose, J. F. and Ross, K. W. (2017). *Computer Networking: A Top-Down Approach*. Pearson, 7th edition.

Lasaro, J. C. (2021). Logical time - distributed systems notes. `https://lasarojc.github.io/ds_notes/time/logical/`. Accessed: June 20, 2023.