Uma Blockchain para Notificação Compulsória de Doenças

Trabalho de Conclusão de Curso

Caio Costa Cavalcante

Allan Edgard Silva Freitas
Orientador

Instituto Federal da Bahia – IFBA
Curso de Análise e Desenvolvimento de Sistemas
Campus Salvador

Salvador, Bahia, Brasil
Dezembro 2023

SUMÁRIO

1. Visão Geral
1.1 Declaração do Problema
1.2 Proposta de Solução de Software
1.2.2 Conceito
1.3 Tecnologias adotadas
1.4 Trabalhos Relacionados
2. Requisitos
2.1 Requisitos Não Funcionais
2.2 Requisitos Funcionais
3. Design
3.1 Projeto UML
3.2 Visão Arquitetural
3.3 Formato de Estrutura de Armazenamento de Dados
4. Implantação
4.1 Projeto de Implantação
5. Manual do Usuário
Agradecimentos
Referências

1. Visão Geral

1.1 Declaração do Problema

Em 2021, após uma tentativa de doação de sangue, o autor deste trabalho foi diagnosticado com a Doença de Chagas. Em seguida, buscou-se informações sobre os notificações e incidências no estado da Bahia e em Salvador, uma vez que a doença é de notificação obrigatória. Foi então identificado o Sistema de Informação de Agravos de Notificação – SINAN (Ministério da Saúde, 2023).

O SINAN, regulamentado pela Portaria 3328/2022 (Diário Oficial da União, 2022), armazena não apenas as notificações compulsórias listadas, mas também pode ser alimentado com agravos provenientes de problemas de saúde de uma região específica (Ministério da Saúde, 2023). Seu propósito é centralizar as notificações por área geográfica, promovendo o uso sistemático e descentralizado. No entanto, ao investigar o registro de casos de Chagas e outros agravos, observou-se falta de uniformidade no registro das notificações (Ministério da Saúde, 2019). Isso claramente expõe um risco de falhas no tratamento dessas informações, dada a manipulação manual de dados sensíveis, sem a segurança proporcionada por sistemas automatizados.

A premissa de descentralização do SINAN permite que qualquer profissional de saúde efetue a notificação. No entanto, dada a heterogeneidade dos sistemas de notificação, surge a dúvida sobre a fidedignidade das notificações que chegam ao sistema em relação à realidade. Ademais, o SINAN foi concebido para analisar os dados das informações registradas como forma de controle das notificações. No entanto, ao tentar extrair dados de casos de Doença de Chagas ocorridos em 2021 na Bahia, não foi identificado nenhum registro.

A motivação subjacente a este trabalho aborda a problemática da centralização, as práticas não padronizadas de registro de agravos no SINAN, a ausência de uma visualização das notificações em um mapa e a falta de transparência na arquitetura e distribuição dos dados do SINAN.

1.2 Proposta de Solução de Software

Desta forma é necessária uma plataforma para dispor tais informações, assegurando distribuição das informações com a segurança entre entidades envolvidas, e mapeamento geográfico dos agravos. O presente trabalho propõe um protótipo como solução, com o escopo correspondente ao Produto Mínimo Viável (MVP) associado ao problema. A utilização é demostrada por meio de uma interface web para o usuário final, que apresenta a localização das notificações e as registra em um sistema distribuído baseado em blockchain, o que permite maior consistência e disponibilidade das informações e favorece a visualização destas.

Blockchain é o registro imutável de transações distribuídas entre os participantes da rede. Na Hyperledger Fabric, a blockchain é composta por um livro-razão (*ledger*) mantida em duas partes principais: o "*World State*" (Estado do Mundo), que reflete o estado atual dos ativos, e o "*Transaction* Log" (Log de Transações), que armazena todas as transações executadas. Cada transação no livro-razão resulta em um conjunto de pares chave-valor que são registrados como criações, atualizações ou exclusões.

1.2.2 Conceito

O conceito da aplicação é um sistema que realiza a notificação de doenças por meio de uma plataforma de blockchain permissionada. Em suma, o software se apresenta por meio de tela única que o usuário cadastra uma notificação no notifica-frontend. A notificação é enviada via http POST para *notifica-backend* através de uma API RESTfull. O *notifica-backend* acessar o contrato inteligente (chaincode) através da interface contract. Através do contract *notifica-backend* pode chamar métodos do *notifica-chaincode* executando em contêineres. Uma vez a notificação chega em notifica-*chaincode* é armazenado na ledger.

Após o envio do POST é feito um GET (sem id especifico) de todas as notificações cadastradas. Todas notificações chegando em *notifica-frontend*, são listadas e plotadas em um mapa dinâmico utilizando o endereço válido passado no cadastro.

1.3 Tecnologias Adotadas

As ferramentas utilizadas na solução proposta foram: plataforma de blockchain hyperledger fabric, golang, javascript, framework reactJS, axios, react-leaflet e API RESTful.

Blockchain

Blockchain é uma tecnologia de registro distribuído que permite a criação de um livro-razão digital compartilhado e imutável. É composto por blocos de dados encadeados em ordem cronológica, onde cada bloco contém um conjunto de transações com mudança de estados. A característica fundamental do blockchain é a descentralização, o que significa que não existe uma autoridade central que controle o sistema (apesar de algumas implementações colocar a figura de um mediador). Em vez disso, a validação das transações é realizada por uma rede de participantes (nós) através de um processo de consenso (Nakamoto, 2008). Blockchain foi escolhido como solução devido a sua descentralização das informações com réplicas e com a segurança do consenso. Sua arquitetura P2P distribui os réplicas dos dados entre os nós, assim, caso um nó caia, a informação estará salva em outro nó.

Além disso, o consenso vem como forma de trazer confiança para redes distribuídas tentando resolver o acordo na tomada de decisão. O que pode ser ilustrado pelo clássico problema dos Generais Bizantinos: Este um cenário hipotético na teoria dos sistemas distribuídos, que descreve a situação em que um exército está cercando uma cidade e precisa coordenar um ataque. No entanto, alguns generais são traidores e podem enviar mensagens falsas, levando a um ataque descoordenado e, consequentemente, à derrota do exército. O desafio é atingir uma tomada de decisão a partir de um quórum mínimo de generais confiáveis, alcançando o consenso distribuído, apesar das falhas (Lamport, 1982).

Essa analogia pode ser aplicada ao ambiente descentralizado de um blockchain. Em um blockchain, existem vários nós (ou participantes) que mantêm uma cópia do livro-razão. Garantir que todos os

nós cheguem a um consenso sobre a validade das transações é crucial. A força da rede blockchain é o interesse da maioria manter a rede íntegra. O consenso distribuído, onde a maioria dos nós deve concordar sobre a validade das transações antes que elas sejam adicionadas na cadeia. Isso é alcançado usando algoritmos, como, por exemplo, o Proof of Work, que foi proposto por Nakamoto (2018).

Algoritmos de consenso

Os algoritmos de consenso são a raiz da blockchain. Eles buscam garantir o maior nível de integridade e confiabilidade das redes. Em suma, algoritmos de consenso resolvem cada um a sua maneira processo de confiança de armazenamento do estado na ledger compartilhado entre os participantes. Diversos algoritmos de consenso foram propostos, cada um com suas características distintas. Alguns exemplos são o Proof of Work de Nakamoto (2018), Proof-of-Stake (PoS) e Practical Byzantine Fault Tolerance (PBFT).

RAFT

O algoritmo RAFT, proposto por Diego Ongaro e John Ousterhout em 2014, é um algoritmo de consenso projetado para sistemas distribuídos (não só blockchains). Sua principal característica é a busca por simplicidade e compreensibilidade, diferenciando-se de algoritmos mais complexos como o PAXOS. O objetivo fundamental do RAFT é oferecer uma solução que seja fácil de entender, implementar e depurar. (Ongaro, 2014).

O funcionamento do Raft é baseado em uma divisão de papéis entre os nós da rede: líder, seguidor e candidato. O processo de eleição é central para o algoritmo, no qual os nós competem para se tornar líderes. O líder é responsável por coordenar e propagar as operações na rede, enquanto os seguidores mantêm uma cópia do estado atual e seguem as instruções do líder. Caso o líder falhe, um novo processo eleitoral é iniciado para eleger um novo líder.(Ongaro, 2014)

O Raft opera em fases distintas, incluindo eleição, replicação de logs e manutenção de consistência. Essas fases são projetadas para garantir que todos os nós na rede concordem sobre a ordem das operações e o estado atual. (Ongaro,2014)

PAXOS

O algoritmo PAXOS, desenvolvido por Leslie Lamport, é um algoritmo aplicado em blockchains e ambientes onde a tolerância a falhas é essencial. Proposto por Lamport em 1998, o PAXOS é conhecido por sua robustez e capacidade de operar em ambientes onde alguns nós podem falhar ou agir de maneira maliciosa.(Lamport, 1998)

O PAXOS funciona através de um conjunto de fases, incluindo proposição, aceitação e aprendizado, que visam garantir que todos os nós cheguem a um consenso sobre ação proposta. Um aspecto fundamental do PAXOS é sua capacidade de lidar com situações onde alguns nós na rede podem estar inativos ou fornecer informações incorretas. O protocolo mantém a consistência e a integridade mesmo em cenários adversos.(Lamport, 1998)

A complexidade do PAXOS reside na compreensão detalhada das fases e na lógica por trás das trocas de mensagens entre os nós. Embora seja reconhecido por sua eficácia, o PAXOS é frequentemente considerado mais difícil de implementar e entender do que alternativas mais recentes, como o RAFT.

Proof-of-Work (PoW)

É um algoritmo de consenso utilizado em blockchain, descrito por Satoshi Nakamoto (idealizador do Bitcoin) em (Nakamoto, 2008). O PoW é projetado para resolver o problema de duplo gasto e garantir a segurança e a integridade da rede descentralizada.

No sistema PoW, os participantes, chamados mineradores, competem para resolver problemas matemáticos complexos. Esse processo é chamado de mineração. O primeiro minerador que consegue resolver o problema é autorizado a adicionar um novo bloco à blockchain e é recompensado com uma quantidade específica de criptomoeda, como o Bitcoin. Resolver o problema requer um poder computacional significativo, o que torna a criação de novos blocos um processo difícil e intensivo em recursos.

O PoW proporciona segurança à blockchain, pois qualquer tentativa de alterar um bloco anterior exigiria a recriação de todos os blocos subsequentes, o que é computacionalmente inviável devido à dificuldade crescente dos problemas. No entanto, o PoW é criticado por seu alto consumo de energia, uma vez que os mineradores competem globalmente para resolver esses problemas complexos. Como resultado, várias blockchains estão explorando alternativas mais sustentáveis, como o Proof-of-Stake (PoS) e outras variantes de consenso.

Practical Byzantine Fault Tolerance (PBFT)

É um algoritmo de consenso desenvolvido para garantir a confiabilidade e segurança em sistemas distribuídos, especialmente em ambientes nos quais alguns participantes podem se comportar de maneira maliciosa. Proposto por Miguel Castro e Barbara Liskov em 1999, o PBFT é projetado para operar em redes assíncronas, onde a comunicação entre os nós não é instantânea.

O PBFT se destaca por sua capacidade de tolerar até um terço dos nós na rede agindo de maneira maliciosa ou falhando, ao mesmo tempo que mantém a consistência e a integridade das transações. O algoritmo opera em etapas, incluindo a fase de pré-preparação, preparação e comprometimento, garantindo que os nós na rede concordem sobre a ordem das operações.(Castro, 1999)

Uma característica importante do PBFT é a redução do número de mensagens necessárias para alcançar o consenso, o que o torna mais eficiente do que alguns outros algoritmos de consenso em ambientes nos quais a latência de rede pode ser uma preocupação. No entanto, o PBFT requer uma comunicação confiável entre os nós, sendo mais adequado para ambientes nos quais a maioria dos participantes são considerados confiáveis.(Castro, 1999)

Proof of Stake (PoS)

É um mecanismo de consenso em blockchain que difere do Proof of Work (PoW) em termos de como os participantes são escolhidos para validar e adicionar novos blocos à cadeia. No PoS, em vez de depender da capacidade de processamento computacional, a seleção é baseada na quantidade de moeda ou recursos financeiros mantidos pelos participantes (stakeholders). Quanto mais moedas um participante possui, maior a probabilidade de ser escolhido para criar um novo bloco. Esse design visa promover a segurança e a participação ativa de quem tem um interesse financeiro no sucesso da blockchain. Além disso, o PoS é elogiado por sua eficiência energética em comparação com o PoW, já que não requer a resolução de problemas computacionais complexos.

Proof of Capacity

É outro mecanismo de consenso, também conhecido como Proof of Space. Ao contrário do PoW e PoS, o PoC não envolve cálculos computacionais intensivos ou detenção de moedas, mas sim a

alocação de espaço de armazenamento. Os participantes demonstram que reservaram uma quantidade significativa de espaço em disco, pré-preenchido com dados randomizados chamados de "plots" no contexto do PoC. A escolha de quem cria o próximo bloco é baseada na capacidade de armazenamento disponível, e não na quantidade de poder computacional ou moeda detida. O PoC é considerado uma alternativa mais sustentável em termos de consumo de energia, pois a mineração é essencialmente feita durante a fase de alocação de espaço, sem a necessidade de constantes cálculos intensivos.

Outros Elementos

Em rede funcional, existe além de consenso também possui "políticas" e algoritmos de disseminação de informações. As políticas são critérios criados juntos com o canal para aprovação de modificações na rede. Por exemplo, regras de ingresso de novos membros ou exclusão são descritas nas políticas.

A politica usada de destaque no presente trabalho é aprovação do deploy do chaincode (é utilizada o operador lógico AND em que todas as organizações precisam aprovar o deploy). Apesar de ser uma eleição simulada, é levada em consideração que mesmo em ambientes simulados, a política deve ser respeitada.

Na imagem demonstra o log de consenso para deploy de chaincode:

```
+ peer lifecycle chaincode checkcommitreadiness --channelID mychannel --name notifica-chaincode --version 1.0.1 --sequenc
e 1 --output json
+ res=0
{
        "approvals": {
                "Org1MSP": true,
                "Org2MSP": true
        }
}
```

Figura 1: Log de saída do deploy do chaincode logo após a execução com o network.sh no diretório de test-netwok

No consenso, o hyperledger utiliza um componente modular de consenso que controla serviços de pedidos de transação. Na versão 2.x em diante é utilizado o serviço de tolerante a falhas de colisão com a biblioteca etcd com protocolo RAFT.

O protocolo Raft, usado na implementação do serviço de pedido no Fabric, segue um modelo de líder e seguidores, onde um líder é escolhido entre os nós de pedido em um canal. Se alguns nós falharem, contanto que a maioria dos nós ainda esteja operante (chamado de "quorum"), o sistema consegue lidar com falhas. Isso o torna tolerante a falhas, permitindo, por exemplo, a perda de um nó em um canal de três nós ou até dois em um canal de cinco (Hyperledger, 2023).

Gossip

O protocolo de disseminação de dados Gossip ("fofoca") é um protocolo que permite que os peers compartilhem informações do livro-razão de forma contínua e escalável. Por meio do Gossip, cada peer recebe e transmite dados constantemente, garantindo que todos possuam informações atualizadas. Além disso, a assinatura de cada mensagem (outorgado pela autoridade certificadora, que normalmente em ambiente de produção é o Fabric CA) impede que dados falsos sejam inseridos na rede, garantindo sua integridade. (Hyperledger, 2023)

O Gossip protocol executa três funções principais: Primeiro, gerencia a descoberta de peers na rede e a associação a canais específicos, garantindo que todos os peers estejam conectados e atualizados. Segundo, dissemina os dados do livro-razão entre todos os peers do canal, permitindo que qualquer peer desatualizado identifique e sincronize blocos ausentes. Por fim, o protocolo também facilita a atualização de novos peers, garantindo que eles recebam os dados necessários

para se integrarem à rede de forma consistente (estrutura arquitetônica de uma Peers-to-Peers). (Hyperledger, 2023)

Existe a eleição de líderes. Esses líderes são responsáveis por iniciar a disseminação de novos blocos entre os peers de suas organizações, otimizando a utilização da largura de banda do serviço de ordenação. Esses líderes podem ser eleitos de maneira estática, com configurações manuais, ou dinâmica, através de um processo de eleição entre os próprios peers da organização. (Hyperledger, 2023)

Além disso, os peers âncoras ao permitem a comunicação entre peers de diferentes organizações. Eles fazem parte da garantia que informações entre organizações sejam compartilhadas de forma eficiente e segura. Por fim, a constante troca de mensagens "alive" e o processo de reconciliação de estado garantem a integridade e a consistência dos dados na blockchain mesmo em situações de falhas ou desconexões temporárias. (Hyperledger, 2023)

No contexto do blockchain, a analogia com o "Problema dos Generais Bizantinos" destaca a importância de um sistema robusto de consenso, de políticas e de disseminação de informação (“fofoca”) para garantir a integridade e a segurança das transações em ambientes descentralizados. Essa é uma das razões pelas quais o blockchain é considerado uma inovação significativa na tecnologia de registros distribuídos.

Hyperledger

Hyperledger é uma iniciativa de código aberto (que estabelece uma transparência na plataforma) liderada pela Linux Foundation, que visa promover a colaboração e o desenvolvimento de tecnologias blockchain empresariais. Diferentemente de algumas outras plataformas blockchain públicas (como Bitcoin (BITCOIN.ORG, 2023) e Ethereum (ETHEREUM.ORG, 2023)), o Hyperledger foca em soluções para empresas e organizações, oferecendo ferramentas e frameworks para desenvolvimento de aplicações blockchain privadas e permissíveis. (Hyperledger, 2023)

Hyperledger Fabric

Hyperledger Fabric é um dos frameworks blockchain sob o guarda-chuva do projeto Hyperledger. Ele é projetado para ser uma plataforma de blockchain modular, permitindo a implementação de soluções personalizadas para casos de uso empresariais. Algumas características que foram exploradas nesse trabalho foram: o controle de privacidade e permissões para que só pessoas autorizadas poderiam fazer uso dos recursos da rede; os smart contracts (chaincodes) que são blocos de códigos que estabelecem lógica nas transações e são ativados de acordo com condições internas ou externas a rede; e suporte a canais: permite a criação de canais privados entre participantes, onde apenas os membros desse canal têm visibilidade das transações. (Hyperledger, 2023)

A “*chain*” no Hyperledger Fabric é um registro sequencial de transações estruturado em blocos conectados por hashes. Cada bloco contém um conjunto de transações que são sequenciadas e criptograficamente ligadas, formando um histórico imutável. Essa cadeia de blocos é armazenada no sistema de arquivos do peer, suportando a natureza de apenas anexar novos dados.

Os chaincodes, também conhecidos como "*smart contracts*", são trechos de código que aguardam execução por algum gatilho (assim como *triggers* em bancos de dados) para fazer operações na blockchain. Eles são implantados na rede para definir a lógica de negócios e interagir com o ledger. Os chaincodes executam transações, atualizando o estado do ledger com base nas regras e lógica

estabelecidas. Os smart contracts determinam automaticamente ações com base em condições pré-definidas, garantindo transparência e confiança nas transações.

Figura 2: Esquema de etapas de efetivação de um chaincode em um canal

O fluxo de transação no Hyperledger Fabric envolve a proposta e verificação da transação por peers endossantes, a execução do chaincode para simular a transação e a validação das transações pelos peers antes da confirmação. Essa validação inclui a verificação da política de endosso, autenticação das assinaturas e verificação do conjunto de leitura para garantir a integridade dos dados.

Na Figura 3 é demonstrado um pseudo código com a interface do chaincode: Duas organizações fazendo transação de compra e venda de carros com o metódo transfer() e com o método update é alterando a cor.

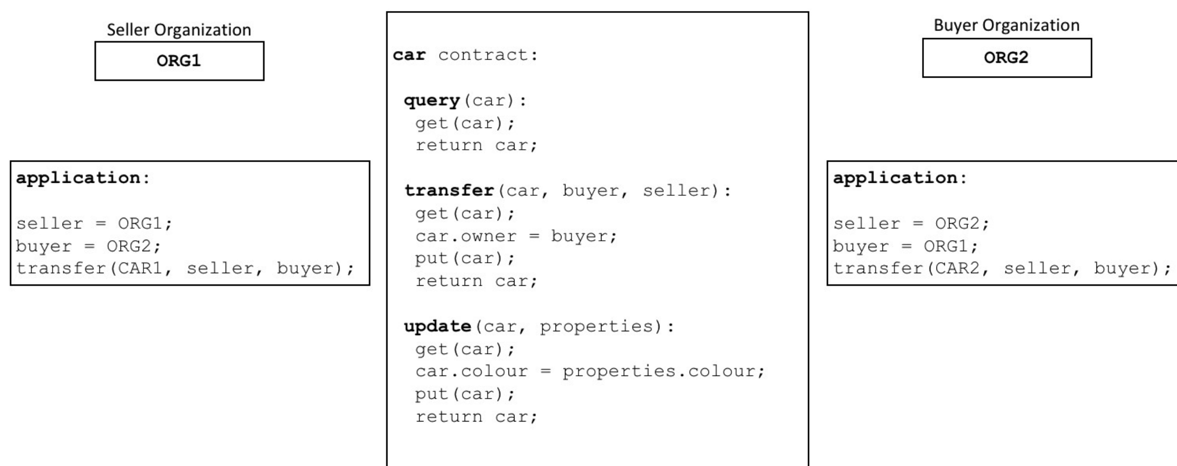

Figura 3: Pseudo código de um chaincode com duas organizações fazendo transação de compra e venda

Trechos de chaincode usado nesse trabalho:

Figura 4 mostra interface utilizada no código deste trabalho: GetUltimoId, InitLedger, CreateAsset, ReadAsset, AssetExists, GetAllAssets. Metódos que podem ser chamados pelo método SubmitTransaction utilizando o padrão de projeto Command.

```
class SmartContract
func GetUltimoId
func InitLedger
func CreateAsset
func ReadAsset
func AssetExists
func GetAllAssets
```

Figura 4: Métodos do chaincode organizados pelo github

Figura 5 mostra struct do SmartContract que agrega uma API pré estabelecida do tipo Contract.

```
type SmartContract struct {
        contractapi.Contract
}
```

Figura 5: Struct do chaincode agregado api do Contract

Figura 6 mostra a implementação do método GetUltimoId(). Método utilizado para monitorar o último id utilizado em um asset.

```
var ultimoId int

//gerarObjetoTeste retorna dois objetos com v

func (s *SmartContract) GetUltimoId() int {
        return ultimoId
}
```

Figura 6: Método GetUltimoId

Figura 7 mostra o método InitLedger() que é utilizado para criar um asset de demonstração.

```
func (s *SmartContract) InitLedger(ctx contractapi.TransactionContextInterface) error {
	asset := notifica_model.Asset{
		Id:                   1,
		DocType:              "notificacao",
		DataNascimento:       "28/06/1988",
		Sexo:                 "Masculino",
		Endereco:             "Av Cardeal Avelar Brandão Villela 6",
		Bairro:               "Jardim Santo Inácio",
		Cidade:               "Salvador",
		Estado:               "Bahia",
		Pais:                 "Brasil",
		Doenca:               "Chagas",
		DataInicioSintomas:   "01/11/2023",
		DataDiagnostico:      "04/11/2023",
		DataNotificacao:      "08/11/2023",
		InformacoesClinicas: "tá ruim",
	}

	aEmBytes, _ := json.Marshal(asset)

	return s.CreateAsset(ctx, string(aEmBytes))

}
```

Figura 7: Implementação do InitLedger criando um Asset de demonstração

Figura 8 mostra o método ReadAsset que é faz a leitura dos assets armazenados na ledger. Ele utiliza da api de contexto de transação chamando o GetStub().GetState() parametrizando com o id do asset. É verificado a ocorrência de erro na busca do estado. E é convertido o asset, que foi recebido em formato JSON, em uma struct do tipo notifica_model.Asset.

```
func (s *SmartContract) ReadAsset(contexto contractapi.TransactionContextInterface, idAsset string) (*notifica_model.Asset, error) {
	assetEmBytes, err := contexto.GetStub().GetState("Asset" + idAsset)

	if err != nil {
		return nil, fmt.Errorf("Falha em consultar em Notificação na Ledger com GetState %s", err.Error())
	}

	if assetEmBytes == nil {
		return nil, fmt.Errorf("Notificacao%s não existe", idAsset)
	}

	asset := new(notifica_model.Asset)
	_ = json.Unmarshal(assetEmBytes, asset)

	return asset, nil
}
```

Figura 8: Implementação do ReadAsset

Figura 9 mostra o método CreateAsset que cria um Asset novo. O método recebe um asset em formato de string. É convertido em um slice de bytes. E logo após é armazenado o estado na

blockchain utilizando o método GetStub().PutState() utilizando Asset + id como id na blockchain e incrementado atributo ultimoId.

```
func (s *SmartContract) CreateAsset(contexto contractapi.TransactionContextInterface, asset string) error {

	/**exists, err := s.AssetExists(contexto, strconv.Itoa(id))
	  if err != nil {
		return err
	  }
	  if exists {
		return fmt.Errorf("the asset %s already exists", id)
	  }
	  **/

	assetEmBytes := []byte(asset)
	var a notifica_model.Asset
	_ = json.Unmarshal(assetEmBytes, &a)

	//Chave do estado é Asset + Id da asset
	//Cuidado para não salvar uma Asset com mesmo Id pois são utilizados para salvar na ledger
	err := contexto.GetStub().PutState("Asset"+strconv.Itoa(a.Id), assetEmBytes)
	if err != nil {
		log.Fatalf("Erro ao salvar na ledger %s", err)
	}
	ultimoId++

	return err
}
```

Figura 9: Implementação do CeateAsset

Figura 10 mostra testagem se existe um asset com o mesmo id na ledger. Então caso ocorra de todos as outras informações sobre o asset foram iguais com exceção do id, ele será armazenado.

```
func (s *SmartContract) AssetExists(contexto contractapi.TransactionContextInterface, idAsset string) (bool, error) {
	assetEmBytes, err := contexto.GetStub().GetState("Asset" + idAsset)
	if err != nil {
		return false, fmt.Errorf("falhou em consultar a existência da Notificacao: %v", err)
	}

	return assetEmBytes != nil, nil
}
```

Figura 10: Método de verificação da existência de Assets com mesmo id

Figura 11 mostra o método GetAllAssets de busca de todos os assets na ledger. Utiliza-se o método GetStub().GetStateByRange que obtém uma coleção de estruturas da ledger. Percorre essa coleção convertendo em structs e armazena em outra coleção e retorna.

```go
func (s *SmartContract) GetAllAssets(ctx contractapi.TransactionContextInterface) ([]*notifica_model.Asset, error) {
	// range query with empty string for startKey and endKey does an
	// open-ended query of all assets in the chaincode namespace.
	resultsIterator, err := ctx.GetStub().GetStateByRange("", "")
	if err != nil {
		return nil, err
	}
	defer resultsIterator.Close()

	var assets []*notifica_model.Asset
	ultimoId = 0
	for resultsIterator.HasNext() {
		ultimoId++
		queryResponse, err := resultsIterator.Next()
		if err != nil {
			return nil, err
		}

		var asset notifica_model.Asset
		err = json.Unmarshal(queryResponse.Value, &asset)
		if err != nil {
			return nil, err
		}
		assets = append(assets, &asset)
	}

	return assets, nil
```

Figura 11: Implementação do GetAllAssets e sua iteação com a coleção retornada pela ledger

React.js (ou React)

React é uma biblioteca JavaScript de código aberto para a construção de interfaces de usuário interativas e dinâmicas em páginas web. Desenvolvido e mantido pelo Facebook, o React permite a criação de componentes reutilizáveis que podem ser compostos para construir interfaces complexas.
A biblioteca se destaca por sua abordagem declarativa, onde os desenvolvedores descrevem como a UI deve ser e o React se encarrega de gerenciar o estado e as atualizações de forma eficiente. (ReactJs.org, 2023). Foi utilizado React pela sua curta curva de aprendizado e praticidade em estabelecer uma interface funcional com poucas ações.

Go (também conhecida como Golang)

Go, ou Golang, é uma linguagem de programação de código aberto desenvolvida pela Google em 2007 e lançada publicamente em 2009. Foi projetada para ser uma linguagem de programação simples, eficiente, concorrente e altamente performática. Go é conhecida por sua sintaxe limpa, compilação rápida e execução eficiente.(Golang.Org, 2023)

Go, uma linguagem de programação, apresenta características distintas de outras linguagens mais clássicas como Java, Python e C++. Ela destaca-se por sua eficiência e desempenho superiores, principalmente pela sua facilidade em lidar com programação paralela com as goroutines que permitem operações assíncronas e paralelas. (Golang.Org, 2023)
Ela é de tipagem estática, o que significa que os tipos de dados são verificados durante o processo de compilação. O que facilita a verificação de erros antes de qualquer execução e deixando erros mais evidentes para o desenvolvedor.

A sintaxe do Go é objetiva e clara dando enfoque a sempre fazer uma operação de uma maneira única sem trazer confusão na leitura de código. Além disso possui semelhanças com linguagem C

que muitos desenvolvedores estudam na faculdade. Possui um bom gerenciamento de módulos que facilita na gestão de dependências.

E por fim, a maior justificativa no uso do go é a linguagem nativa do código do Hyperledger Fabric, o que facilita a documentação da comunidade de desenvolvedores e há facilidade na transcriação do código para a execução da plataforma.

1.4 Trabalhos Relacionados

Preethi,2021

Propõe utilização de modelo de três organizações básicas, sendo que uma das organizações seria a gestora da entrada de novas organizações. Utiliza Hyperledger Fabric assim como esse trabalho.A solução apresentada trabalha com Hyperledger Composer. Isso implica em limitação e falta de suporte à tecnologia pois em 29 de agosto de 2019 a ferramenta Composer foi descontinuada permanecendo preso a versão do Hyperledger Fabric 1.4. A solução deste trabalho é com Hyperledger Fabric 2.5, sendo que essa é a versão de suporte de longo prazo (em inglês long-term support), de sigla LTS).

É utilizado NodeJS para interação com chaincode, enquanto este faz uso de golang.O trabalho citado apresenta resgate de registro de doenças anteriores diminuindo o tempo de anamnese . Em comparação com esse trabalho, apesar de ser proposta de modificação futura, o escopo corresponde à notificações e não mapeamento de doenças anteriores dos pacientes. Se o paciente possui comorbidades, será o mesmo que deverá informar ao profissional de saúde da unidade. O trabalho citado demonstra a importância de um prontuário eletrônico no atendimento.

Rimsan,2020

O trabalho citado se propõe monitorar pessoas infectadas ou testadas globalmente utilizando de uma estrutura de compartilhamento de dados entre países. Sugere que utilização de blockchain oferece mais segurança de dados estabelecendo que cada país pode adequar suas regras de manipulação de dados. O presente trabalho não pretende monitorar as pessoas, mas sim as notificações que as pessoas provocam no sistema e como a gestão delas influenciam nas pessoas interessadas em dados de doenças notificáveis.

A tecnologia utilizada é a Ethereum com utilização dos "contratos inteligentes". Cada país em postos estratégicos teriam máquinas virtuais executando Ethereum. Usa-se uma interface de programação de aplicativos (original em inglê Application Programming Interface, da sigla API) com o padrão RESTful, assim como esse trabalho que faz uso dos mesmos. Foi utilizado React e Web3, ambas bibliotecas que trabalham com linguagem JavaScript.

O problema de redes baseadas em Ethereum é que as transações dependem de ter um envolvimento financeiro. No caso do trabalho citado, é obtido 3 ethers pelo ambiente de teste, mas é questionável o quanto um ambiente pensado em nativamente em criptomoedas oferece de flexibilidade para transacionar outros tipos de ativos.

Souza, 2021

Este trabalho é pensado em notificação dos casos de sífilis e foi a inspiração para o presente trabalho iniciar pesquisas. Ele utiliza como base a plataforma Ethereum utilizando como base formulário disponibilizado pelo Sistema de Informação de Agravos de Notificação (SINAN). O presente trabalho faz uso de formulário com informações relevantes para qualquer doença notificável.

O Ethereum possui linguagem própria para codificação dos contratos inteligentes chama Solidity enquanto Hyperledger possui Golang, JavaScript, TypeScript e Java. Para interação com o contrato inteligente foi usado Web3, enquanto no presente trabalho foi usado Golang devido sua simplicidade e recursos modernos.

Para interação com o sistema de notificação de sífilis, foi usado famework web Django, equante o presente trabalho usa bibliotecas Gorilla/Mux e Negroni.

Nem desses trabalhos possuem plotagem de mapa dinâmico web das notificações feitas.

Trabalho	**Modelo Proposto**	**Preethi**	**Rimsen**	**Souza**
Plataforma usada	Fabric 2.5	Fabric 1.4	Ethereum	Ethereum
Linguagem desenvolvida cliente	Golang, Node.JS	Node.Js	Node.Js	Python
Dependência de criptomoeda	Não depende de criptomoeda	Não depende de criptomoeda	Depende de criptomoeda	Depende de criptomoeda
Foco das notificações	Notificações de doenças obrigatórias	Sem doença específica	Notificações COVID19	Sífilis
Instituição de foco	Qualquer instituição de saúde do Brasil	Qualquer instituição médica	Global	Não citado
Linguagem do chaincode	Golang chaincodes	Javascript	Solidity	Solidity
Disponibilidade de Mapa	Presente	Ausente	Ausente	Ausente
API de acesso	API REST	Hyperledger Composer	Interação utilizando Web3	Interação utilizando Web3

2. Requisitos

2.1 Requisitos Funcionais

[RF1] Como um desenvolvedor de sistema, eu preciso obter dados das notificações por uma API padrão RESTful;
[RF2] Como profissional de saúde preciso cadastrar notificações de agravos;
[RF3] Como profissional de saúde preciso listar todas notificações de agravos;
[RF4] Como profissional de saúde preciso visualizar notificações de agravos com pontos em um mapa para visualização maior de incidência geográfica;

2.2 Requisitos Não-Funcionais

[RNF1] O sistema deve ser acessível nas plataformas webs;
[RNF2] As informações de notificações deve ser obtidas via API padrão RESTFul;
[RNF3] As notificações devem ser armazenadas na plataforma blockchian Hyperledger Fabric 2.X;
[RNF4] O código do chaincode deve ser em golang;
[RNF5] O código do cliente do chaincode deve ser em golang;

3. Design

3.1 Projeto UML

Diagrama de Componentes:
Componente notifica-frontend utilizando https se conecta com o componente notifica-backend que faz uso do componente notifica-model criar e utilizar objetos. O notifica-backend utiliza-se da api contract utilizando credenciais interage com notifica-chaincode que utiliza o ctx (context) para interagir com a ledger.

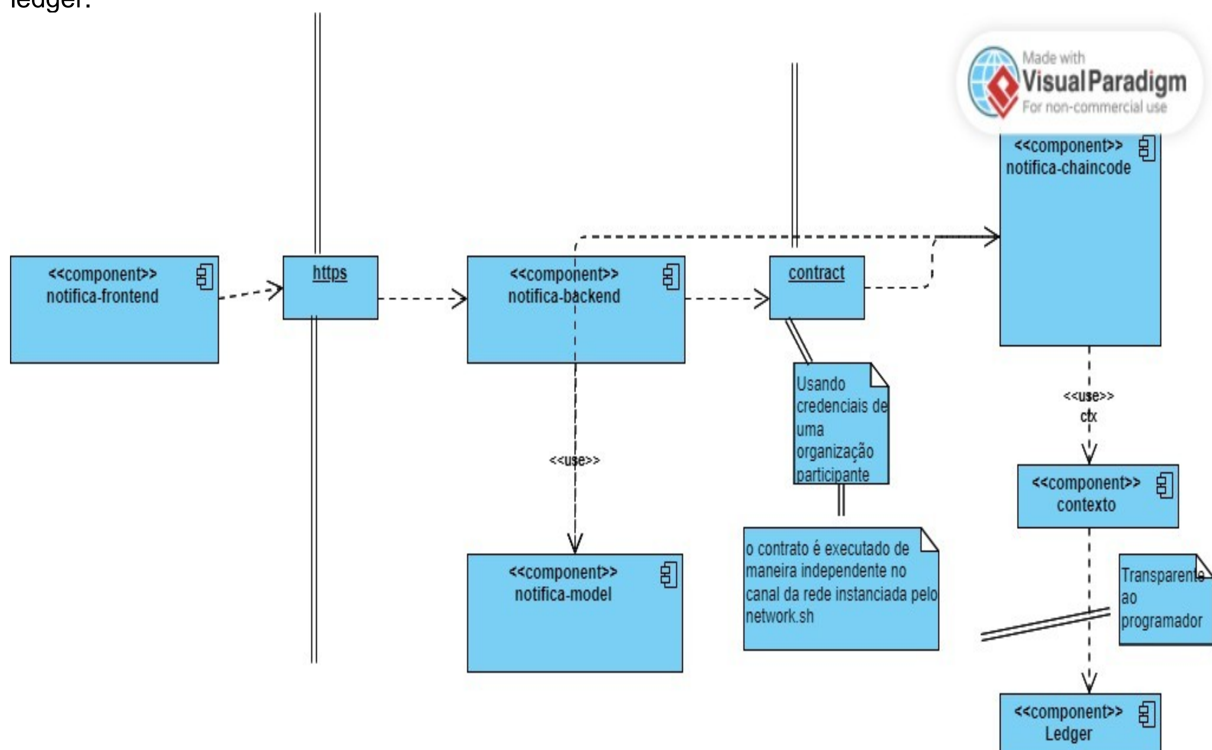

Figura 12: Diagrama de Componentes

Diagrama de Classes
notifica-backend possui um arquivo chamado invoke.go que estabelece uma implementação de uma interface de Organização. Essa interace é um handler de execução do endpoints da API RESTfull que utiliza da estrutura Asset que é a notificação. O objeto contract intermedia a interação do invoke.go com chaincode do notifica-chaincode.

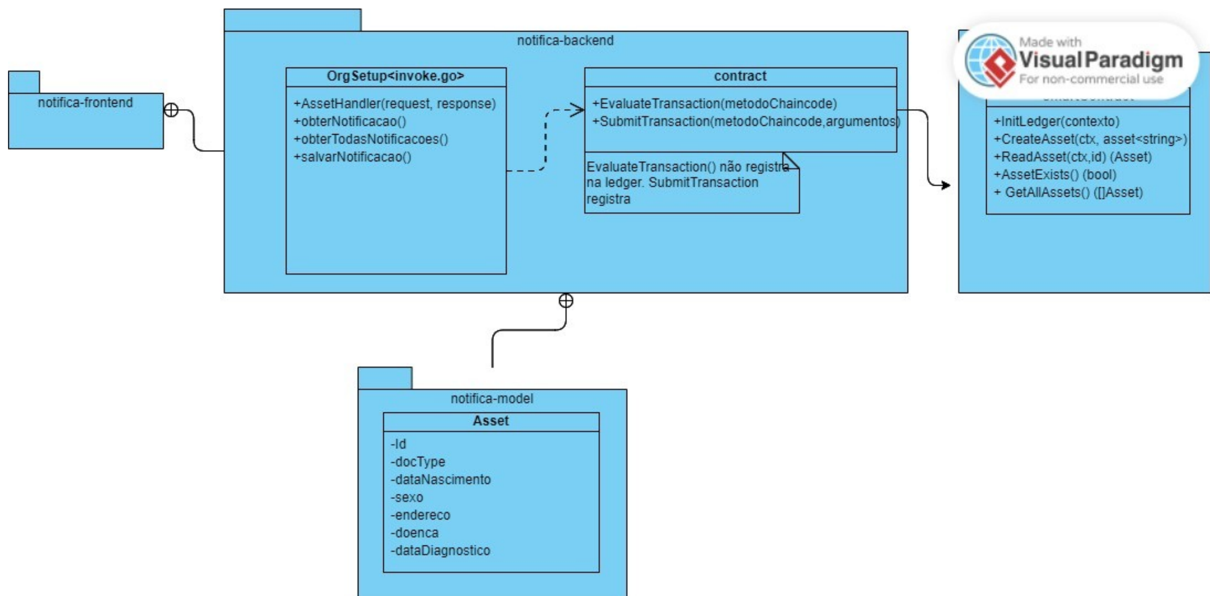

Figura 13: Diagrama de Classes

Diagrama de Implantação

Servidor Web executando em uma máquina qualquer com implementação do notifica-frontend executando interage com servidor remoto em qualquer máquina que armazena notifica-backend, notifica-model e notifica-chaincode

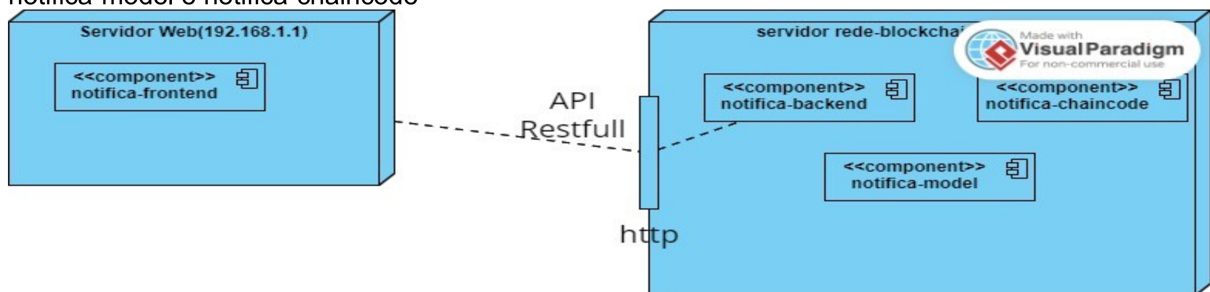

Figura 14: Diagrama de Implantação

Diagrama de Atividade no invoke.go

O arquivo invoke.go possui método "mãe" que mapeia pontos de interação do usuário da API RESTFull. Utiliza-se do GET e id, do GET sem parâmetros e POST (métodos do HTTP). Respectivamente eles utilizam de outros métodos para execução da obtenção dos assets. Utilizam método SumitTransaction como padrão Command para execução de métodos do chaincode.

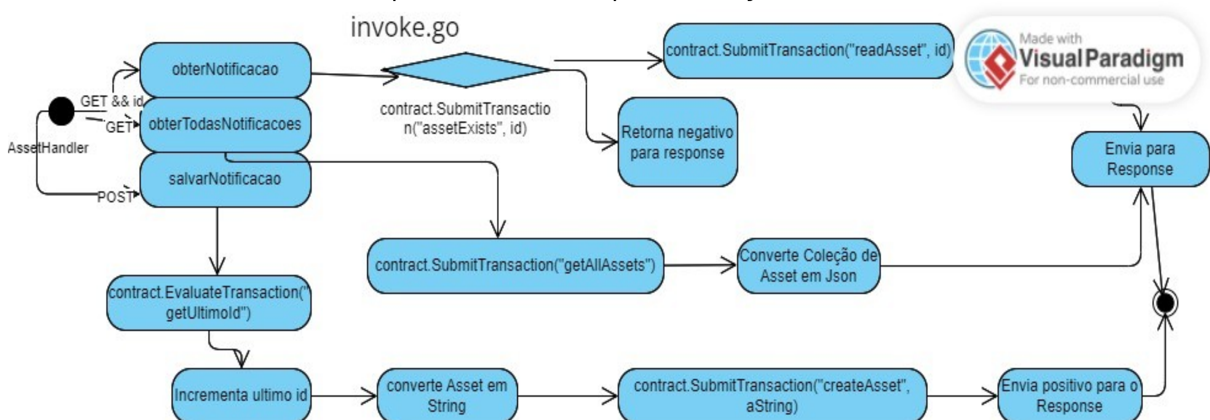

Figura 15: Diagrama de Atividade do invoke.go

Diagrama de Atividade de SmartContract

De acordo com método escolhido pelo cliente do chaincode, os métodos CreateAsset, ReadAsset e GetAllAssets podem ser executados. Pode ser escolhido PutState e GetState, respectivamente o métodos de salvar estado e buscar estado.

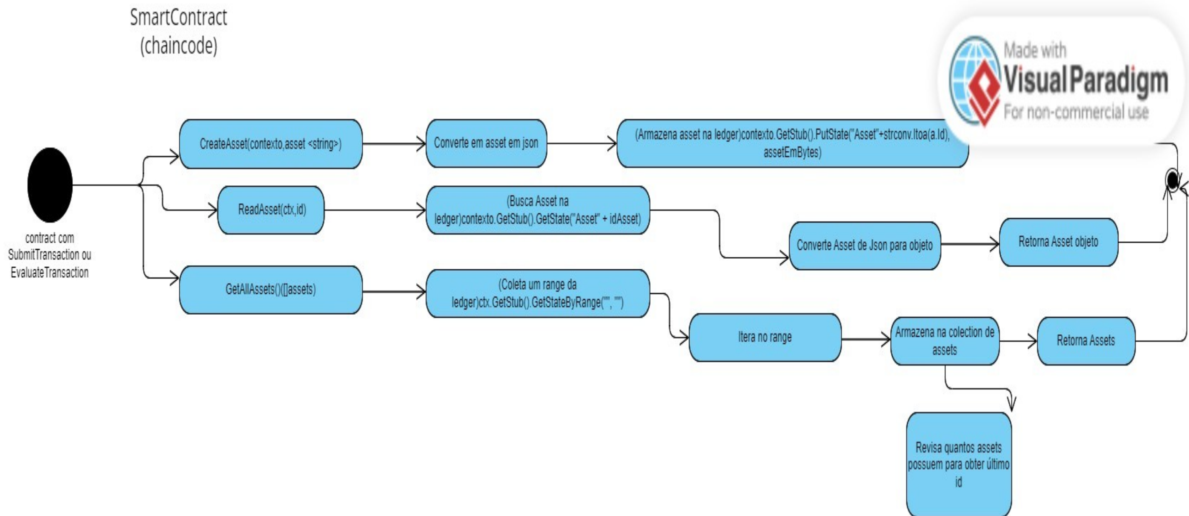

Figura 16: Diagrama de Atividade do SmartContract

Diagrama de Sequência HTTP GET (/notificacao/)

Usuário solicita uma conexão http com notifica-frontend utilizando a porta 3000, assim notifica-frontend solicita através da biblioteca Axios(GET) executa handlers de notifica-backend que por sua vez utiliza contract que interage com chaincodes que altera a ledger.

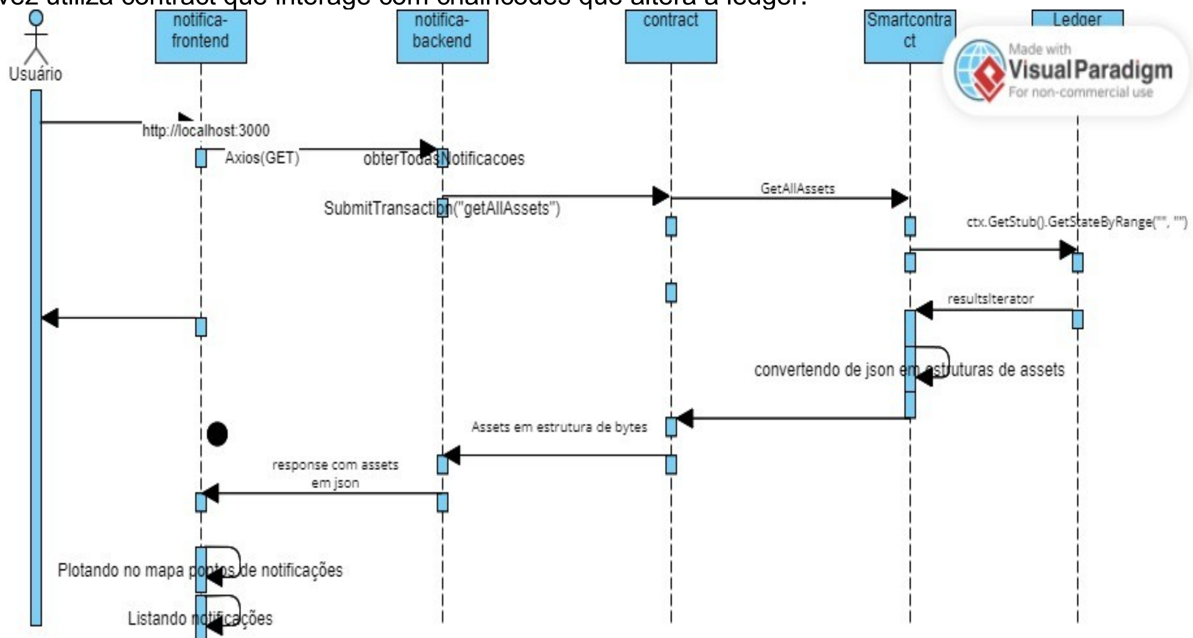

Figura 17: Diagrama de Sequência HTTP GET (/notificacao/)

Diagrama de Sequência HTTP POST
Igual ao método HTTP GET, mas obtém o asset e utiliza do método CreateAsset do notifica-chaincode e também usa o PutState.

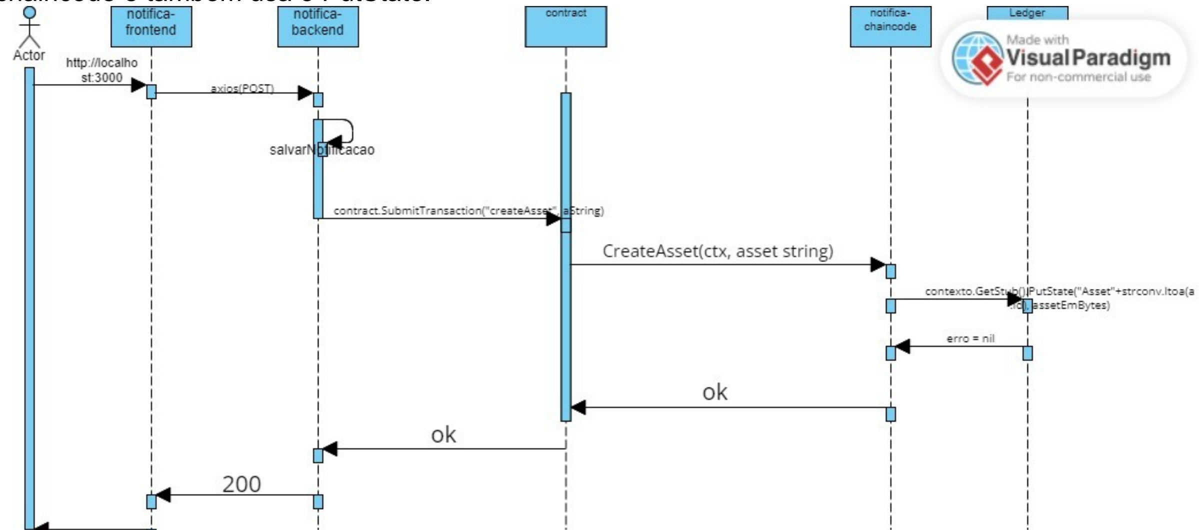

Figura 18: Diagrama de Sequência HTTP POST

Diagrama de Casos de Uso
Usuário pode Cadastrar Notificação, Listar Todas as Notificações e Plotar as Notificações em um Mapa

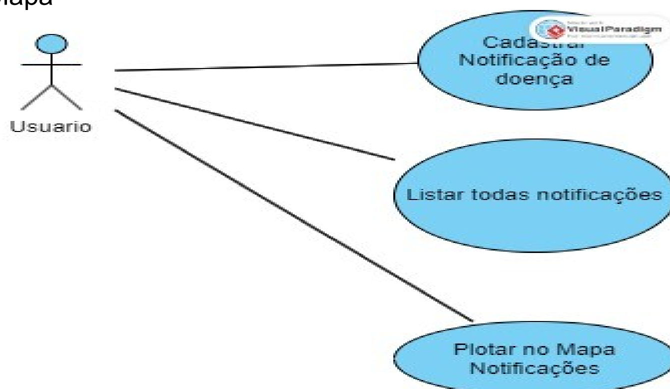

Figura 19: Diagrama de Casos de Uso

3.2 Visão Arquitetural

O principal estilo arquitetural do sistema é Peer-to-Peer pois a ledger é distribuída em todos os nós (peers)(em paralelo a relação entre notifica-frontend e notifica-backend é de Cliente-Servidor). Apesar do sistema desenvolvido está em modelo simples de duas organizações participantes somente, o software desenvolvido, com exceção da implementação do contract (representado no arquivo initialize.go em notifica-backend), todo resto do sistema funciona com uma ou mais organizações participantes pois a interação com a ledger é transparente para o desenvolvedor de chaincodes, principalmente quando ele utiliza o contexto.GetStub().PutState().
Em um sistema de produção, o haveria um orquestrador de conteiners para administrar os diversos peers e orderers(serviço organizador de transações da ledger).

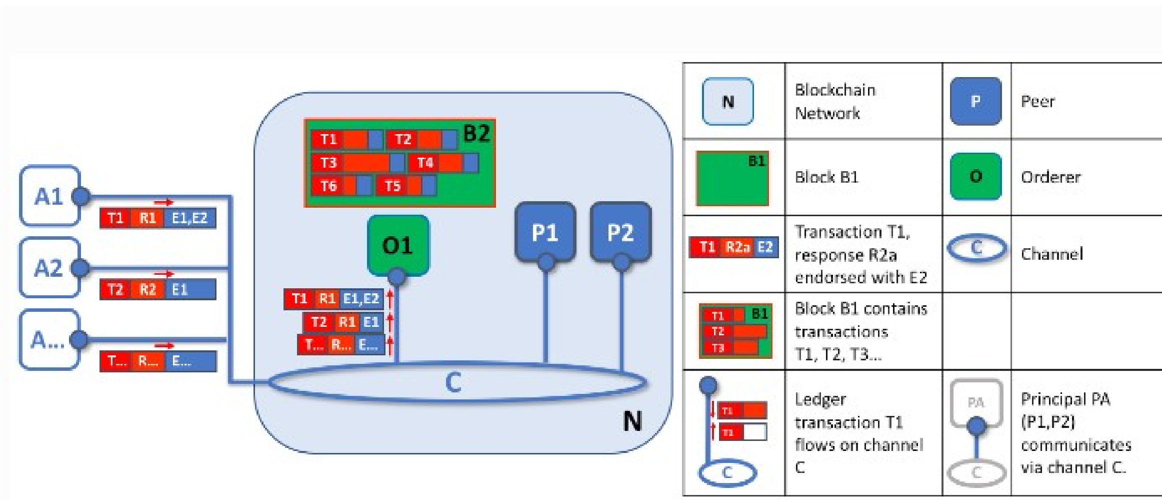

Figura 20: Etapa 1 de solicitação de armazenamento de informações na ledger e como se comporta ordenadores

Na figura 20, primeira função de um nó ordenador é empacotar as atualizações propostas para a ledger. Neste exemplo, a aplicação A1 envia uma transação T1 endossada por E1 e E2 ao orderer O1. Paralelamente, a Aplicação A2 envia a transação T2 endossada por E1 ao orderer O1. O1 empacota a transação T1 do aplicativo A1 e a transação T2 do aplicativo A2 junto com outras transações de outros aplicativos no bloco B2. Podemos ver que em B2 a ordem da transação é T1,T2,T3,T4,T6,T5 – que pode não ser a ordem em que essas transações chegaram ao ordenante! (Este exemplo mostra uma configuração de serviço de solicitação muito simplificada com apenas um nó de solicitação.)

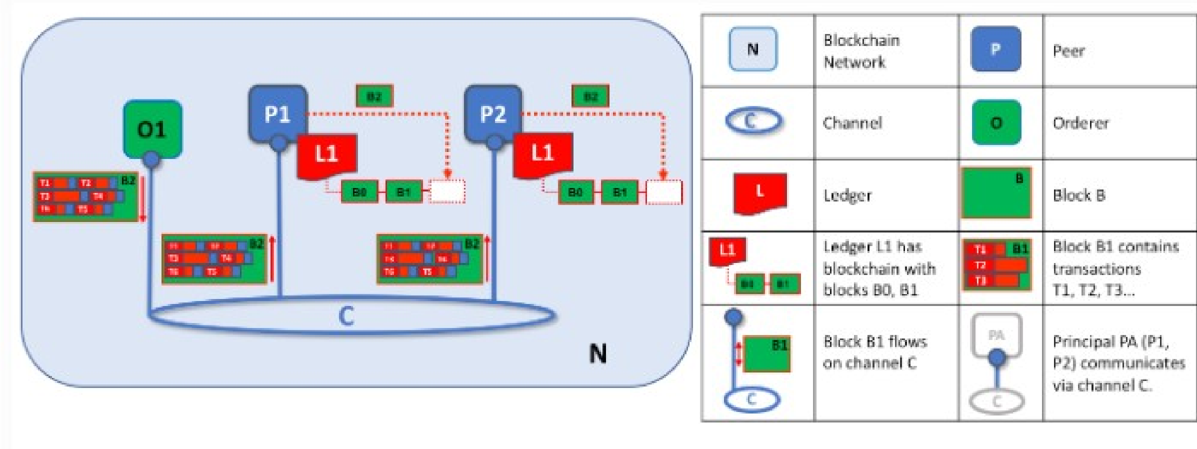

Figura 21: Etapa 2 de distribuição de blocos entre os peers

Na figura 21, segunda função de um nó orderer é distribuir blocos aos peers. Neste exemplo, o ordenador O1 distribui o bloco B2 para o peer P1 e o peer P2. O peer P1 processa o bloco B2, resultando na adição de um novo bloco a ledger L1 em P1. Paralelamente, o par P2 processa o bloco B2, resultando na adição de um novo bloco a ledger L1 em P2. Uma vez concluído este processo, o livro-razão L1 foi atualizado consistentemente nos pares P1 e P2, e cada um pode informar aos aplicativos conectados que a transação foi processada (demonstração encontrada em HYPERLEDGER, 2023).

Com rabanetes

O cenário apresenta dois clientes, A e B, negociando rabanetes, cada um com um peer na rede por meio do qual enviam suas transações e interagem com o livro-razão. (Hyperledger, 2023)

O fluxo assume que um canal está configurado e em funcionamento. O usuário do aplicativo se registrou e se inscreveu na Autoridade Certificadora (CA, normalmente em ambiente de produção com FABRIC CA) da organização, recebendo material criptográfico necessário para autenticar-se na rede. O contrato inteligente (chaincode), contendo pares de chave-valor representando o estado inicial do mercado de rabanetes, está instalado nos peers e implantado no canal. Esse chaincode possui lógica definindo instruções de transação e o preço acordado para um rabanete, além de uma política de endosso exigindo endossos de ambos os peers, peerA e peerB, para qualquer transação. (Hyperledger, 2023)

O Cliente A inicia uma transação solicitando a compra de rabanetes. A solicitação é direcionada para peerA e peerB, representantes dos Clientes A e B, de acordo com a política de endosso. Uma proposta de transação é então construída por meio de um aplicativo que usa um SDK suportado (Node, Java, Golang, no caso desse trabalho em Golang) para gerar a proposta. O SDK formata a proposta utilizando protocol buffer sobre gRPC e usa as credenciais criptográficas do usuário para gerar uma assinatura única. Essa proposta é enviada para um peer alvo (peers âncoras), que encaminha a proposta para outros peers para execução, conforme exigido pela política de endosso. (Hyperledger, 2023)

Os peers que endossam verificam se a proposta de transação está correta, não foi submetida anteriormente, se a assinatura é válida e se o submetente (Cliente A, no exemplo) está autorizado a realizar a operação proposta no canal. Os peers executam o chaincode para produzir resultados de transação, como valores de resposta.. Esses valores, junto com a assinatura dos peers endossantes, são retornados como uma "resposta de proposta" para o peer alvo. Após verificar as respostas de proposta, o peer alvo cria uma transação contendo o ID do Canal, os conjuntos de leitura/escrita e assinaturas dos peers endossantes, enviando esta transação para o serviço de ordenação (orderers). (Hyperledger, 2023)

Os blocos de transações são "entregues" a todos os peers no canal e validados para garantir o cumprimento da política de endosso, além de verificar se não houve alterações no estado do livro-razão desde que o conjunto de leitura foi gerado pela execução da transação. Os peers anexam o bloco ao registro do canal, e para cada transação válida, os conjuntos de escrita são aplicados ao banco de dados de estado atual. Cada peer emite um evento para notificar a aplicação cliente sobre a imutável adição da transação à cadeia, juntamente com a validação ou invalidação da transação. É recomendado que as aplicações escutem o evento da transação para saber se foi ordenada, validada e confirmada no livro-razão. (Hyperledger, 2023)

3.3 Formato de Estrutura de Armazenamento dos Dados

Exemplo de estrutura de um Asset/Notificação na blockchain em JSON:


```
[{"id":1,
"docType":"notificacao",
"dataNascimento":"28/11/1988"
,"sexo":"masculino"
,"endereco":"Rua Emídio dos Santos 6",
"bairro":"Barbalho"
,"cidade":"Salvador",
"estado":"Bahia",
"pais":"Brasil",
"doenca":"COVID19",
"dataInicioSintomas":"10/11/2023",
```

"dataDiagnostico":"02/11/2023",
"dataNotificacao":"10/11/2023",
"informacoesClinicas":"Febre alta"}]

4. Implantação

4.1 Projeto de Implantação

(O notifica-backend, notifica-models, notifica-chaincode e o Hyperledger Fabric foram desenvolvidos em plataforma Linux Ubuntu 20.04)
Instalar os seguintes softwares e versões mais recentes(com exceção do Go):

- git
- curl
- docker
- docker compose
- go 1.19
- jq

Caso queira interagir usando terminal Git Bash no Windows para interação com API REST do notifica-backend faça os seguintes comandos para reconhecimento de quebra de linha em Linux e tratamento e de endereços:
git config --global core.autocrlf false
git config --global core.longpaths true

No terminal do Linux onde estarão todos componentes, exceto notifica-frontend, execute comandos para configuração do golang:
export GOPATH=$HOME/go

export PATH=$PATH:$GOPATH/bin

mkdir -p $HOME/go/src/github.com/<usuário_no_github>
cd $HOME/go/src/github.com/<usuário_no_github>

Para instalar Hyperledger Fabric:

curl -sSLO https://raw.githubusercontent.com/hyperledger/fabric/main/scripts/install-fabric.sh && chmod +x install-fabric.sh
./install-fabric.sh

Para colocar visível ao sistema os binários do hyperledger, no arquivo /home/seuusuario/.bashrc, no final coloque:

export FABRIC_RAIZ=~/go/src/github.com/<usuariogithub>/fabric-samples/

export FABRIC_CFG_PATH=$FABRIC_RAIZ/config/

export export PATH=$FABRIC_RAIZ/bin:$PATH

export CORE_PEER_TLS_ENABLED=true
export CORE_PEER_LOCALMSPID="Org1MSP"

export
CORE_PEER_TLS_ROOTCERT_FILE=$FABRIC_RAIZ/organizations/peerOrganizations/
org1.example.com/peers/peer0.org1.example.com/tls/ca.crt
export CORE_PEER_MSPCONFIGPATH=$FABRIC_RAIZ/organizations/peerOrganizations/
org1.example.com/users/Admin@org1.example.com/msp
export CORE_PEER_ADDRESS=localhost:7051

saia do seu usuário e faça login novamente.

No terminal execute:
cd $FABRIC_RAIZ
cd ..

faça um git clone do model:

git clone https://github.com/Nextc3/notifica-model
cd fabric-samples
cd asset-transfer-basic

faça git clone do chaincode e models
git clone https://github.com/Nextc3/notifica-backend
git clone https://github.com/Nextc3/notifica-chaincode

entre no diretório da rede de test:
cd $FABRIC_RAIZ/test-network

Levante a rede com couchdb, crie um canal com nome padrão:
./network.sh up -s couchdb

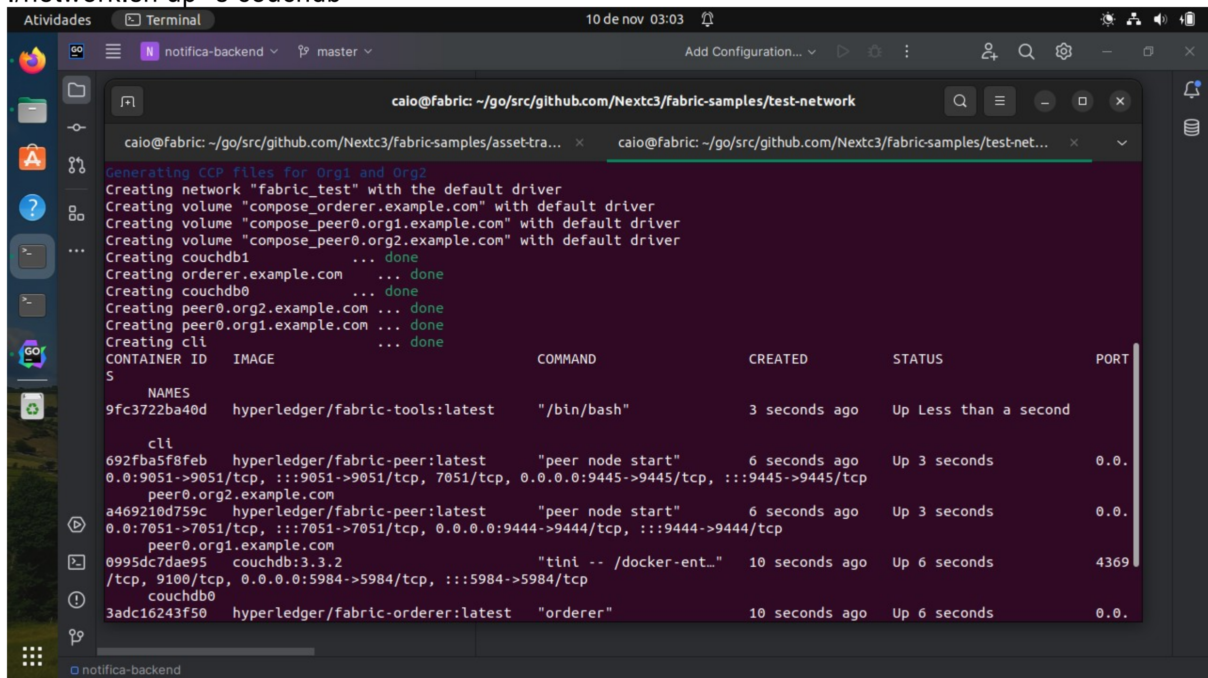

Figura 22: Retorno positivo do log de execução do comando ./network.sh up -s couchdb em executar a rede

./network.sh createChannel

Figura 23: : Retorno positivo do log de execução do comando ./network.sh createChannel na criação do canal

Faça deploy do chaincode

./network.sh deployCC -ccn notifica-chaincode -ccp ../asset-transfer-basic/notifica-chaincode -ccl go

Figura 24: Log de retorno positivo demonstrando a definição de um chaincode sendo feito o deploy no canal

```
Atividades   Terminal   10 de nov 03:11
caio@fabric: ~/go/src/github.com/Nextc3/fabric-samples/test-network
caio@fabric: ~/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic   caio@fabric: ~/go/src/github.com/Nextc3/fabric-samples/test-network
Checking the commit readiness of the chaincode definition successful on peer0.org2 on channel 'mychannel'
Using organization 1
Using organization 2
+ peer lifecycle chaincode commit -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile /home/caio/go/src/github.com/Nextc3/fabric-samples/test-network/organizations/ordererOrganizations/example.com/tlsca/tlsca.example.com-cert.pem --channelID mychannel --name notifica-chaincode --peerAddresses localhost:7051 --tlsRootCertFiles /home/caio/go/src/github.com/Nextc3/fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/tlsca/tlsca.org1.example.com-cert.pem --peerAddresses localhost:9051 --tlsRootCertFiles /home/caio/go/src/github.com/Nextc3/fabric-samples/test-network/organizations/peerOrganizations/org2.example.com/tlsca/tlsca.org2.example.com-cert.pem --version 1.0.1 --sequence 1
+ res=0
2023-11-10 03:10:50.969 -03 0001 INFO [chaincodeCmd] ClientWait -> txid [e124d9ddf48ae375fbc313ecb988a5aaef7d49139308ec08dde5d9d6f6e104f2] committed with status (VALID) at localhost:7051
2023-11-10 03:10:51.011 -03 0002 INFO [chaincodeCmd] ClientWait -> txid [e124d9ddf48ae375fbc313ecb988a5aaef7d49139308ec08dde5d9d6f6e104f2] committed with status (VALID) at localhost:9051
Chaincode definition committed on channel 'mychannel'
Using organization 1
Querying chaincode definition on peer0.org1 on channel 'mychannel'...
Attempting to Query committed status on peer0.org1, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name notifica-chaincode
+ res=0
Committed chaincode definition for chaincode 'notifica-chaincode' on channel 'mychannel':
Version: 1.0.1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org1 on channel 'mychannel'
Using organization 2
Querying chaincode definition on peer0.org2 on channel 'mychannel'...
Attempting to Query committed status on peer0.org2, Retry after 3 seconds.
+ peer lifecycle chaincode querycommitted --channelID mychannel --name notifica-chaincode
+ res=0
Committed chaincode definition for chaincode 'notifica-chaincode' on channel 'mychannel':
Version: 1.0.1, Sequence: 1, Endorsement Plugin: escc, Validation Plugin: vscc, Approvals: [Org1MSP: true, Org2MSP: true]
Query chaincode definition successful on peer0.org2 on channel 'mychannel'
Chaincode initialization is not required
caio@fabric:~/go/src/github.com/Nextc3/fabric-samples/test-network$
```

Figura 25: Log final de deploy do chaincode no canal retornando que tudo ocorreu de maneira correta

vá na pasta ../fabric-samples/test-network/organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/signcerts/ e copie o arquivo existente com extensão .pem e coloque o nome cert.pem sem apagar o original

vá na pasta do backend:
cd $FABRIC_RAIZ/asset-transfer-basic/notifica-backend

execute

go run main.go
Exemplo de quando a execução do backend ocorre errado e logo depois ocorre corretamente após a criação do arquivo cert.pem:

```
Atividades   Terminal   10 de nov 03:13
Navegador Web Firefox   caio@fabric: ~/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic/notifica-backend
caio@fabric: ~/go/src/github.com/Nextc3/fabric-samples/asset-transfer-bas...   caio@fabric: ~/go/src/github.com/Nextc3/fabric-samples/test-network
remote: Compressing objects: 100% (44/44), done.
remote: Total 62 (delta 28), reused 46 (delta 17), pack-reused 0
Receiving objects: 100% (62/62), 66.44 KiB | 1.58 MiB/s, done.
Resolving deltas: 100% (28/28), done.
caio@fabric:~/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic$ git clone https://github.com/Nextc3/notifica-backend
Cloning into 'notifica-backend'...
remote: Enumerating objects: 56, done.
remote: Counting objects: 100% (56/56), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 56 (delta 27), reused 46 (delta 17), pack-reused 0
Receiving objects: 100% (56/56), 15.28 KiB | 173.00 KiB/s, done.
Resolving deltas: 100% (27/27), done.
caio@fabric:~/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic$ cd notifica-backend/
caio@fabric:~/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic/notifica-backend$ go run main.go
2023/11/10 03:12:27 Initializing connection for Org1...
panic: failed to read certificate file: open ../../test-network/organizations/peerOrganizations/org1.example.com/users/User1@org1.example.com/msp/signcerts/cert.pem: no such file or directory

goroutine 1 [running]:
github.com/Nextc3/fabric-samples/asset-transfer-basic/notifica-backend/web.OrgSetup.newIdentity({{0x9d6699, 0x4}, {0x9d8b3e, 0x7}, {0x0, 0x0}, {0xc0001e2280, 0x77}, {0xc00025a070, 0x6e}, ...})
        /home/caio/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic/notifica-backend/web/initialize.go:64 +0x74
github.com/Nextc3/fabric-samples/asset-transfer-basic/notifica-backend/web.Initialize({{0x9d6699, 0x4}, {0x9d8b3e, 0x7}, {0x0, 0x0}, {0xc0001e2280, 0x77}, {0xc00025a070, 0x6e}, ...})
        /home/caio/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic/notifica-backend/web/initialize.go:21 +0x165
main.main()
        /home/caio/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic/notifica-backend/main.go:22 +0x118
exit status 2
caio@fabric:~/go/src/github.com/Nextc3/fabric-samples/asset-transfer-basic/notifica-backend$ go run main.go
2023/11/10 03:12:59 Initializing connection for Org1...
2023/11/10 03:12:59 Initialization complete
Escutando (http://localhost:8080/)...
```

Figura 26: Clonando o repositório e executando o notifica-backend com rede e chaincode já operantes

se tudo estiver ok um servidor web será criado escutando na porta 8080.

Teste se o chaincode funciona e execute dentro da pasta $FABRIC_RAIZ/test-network:

peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.example.com --tls --cafile "${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem" -C mychannel -n notifica-frontend --peerAddresses localhost:7051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt" --peerAddresses localhost:9051 --tlsRootCertFiles "${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt" -c '{"function":"InitLedger","Args":[]}'

Se ocorrer certo retornará:
INFO 001 Chaincode invoke successful. result: status:200

Faça uma consulta ao chaincode com:
peer chaincode query -C mychannel -n notifica-chaincode -c '{"Args":["GetAllAssets"]}'

retornará asset padrão criado com método InitLedger

Em outro computador baixe o frontend:

git clone https://github.com/Nextc3/notifica-frontend

tenha instalado Node.JS, npm, React.JS

Com todos instalados entre dentro de notifica-frontend e execute: npm start

No arquivo: notifica-frontend\src\main\App.js na linha 27 coloque o ip onde o notifica-backend está executando:
const baseUrl = "http://192.168.1.80:8080/notificacao/"

5. Manual do Usuário

Caso tudo ocorra bem na implantação, o React.JS irá abrir o navegador e ir exibir a página de cadastro de notificação no http:localhost:3000.
A página de cadastro é semelhante as imagens listadas abaixo.
Primeiro aparece o formulário de preenchimento de uma Notificação, abaixo aparecerá o mapa dinâmico para plotagem das notificações já inseridas e mais abaixo aparecerá lista de notificações.
Primeiro preencha o formulário cadastrando alguma notificação:

Figura 27: Imagem superior do cadastro web de notifica-frontend recebendo alguns dados

Ao final escolha a doença notificável:

Figura 28: Parte 2 do cadastro de notificação demonstrando escolha das doenças que podem ser notificadas

Após o envio das informações aparecerá um pino no mapa indicando o endereço da notificação:

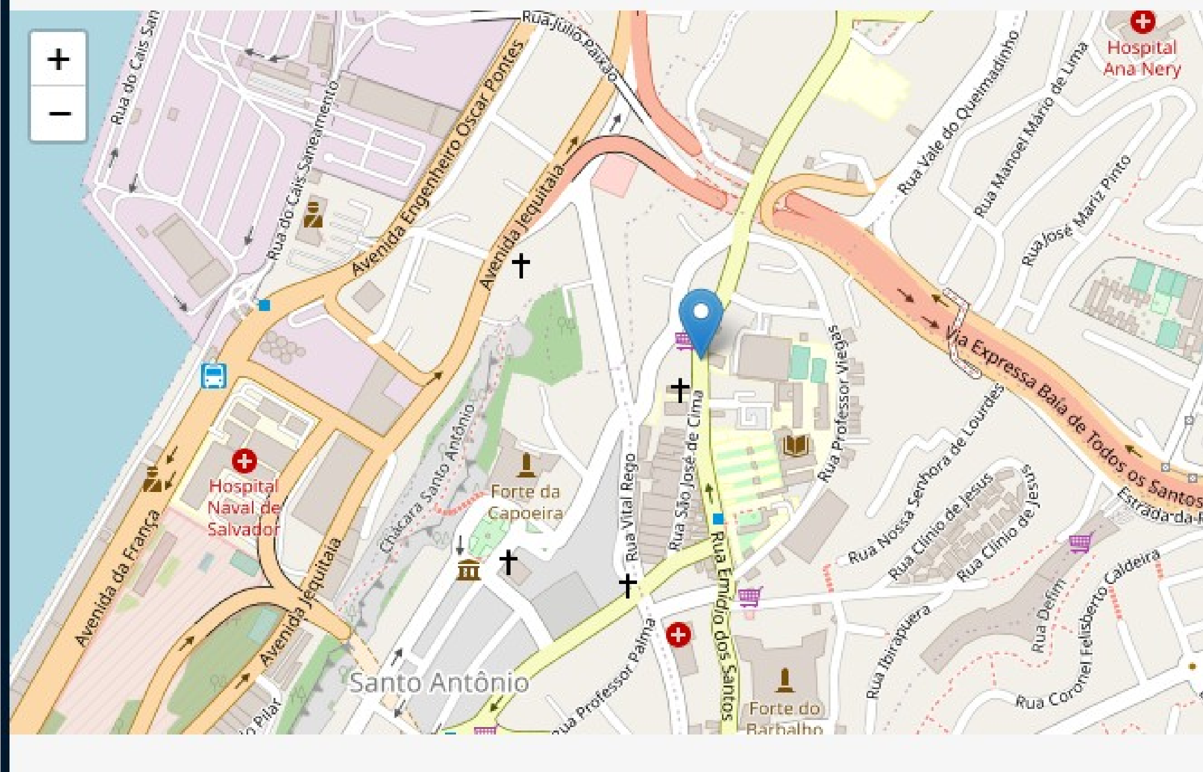

Figura 29: Mapa dinâmico mostrando “alfinete” em um bairro cadastrado da cidade de Salvador, Bahia e Brasil

Clicando no pino aparecerá informações sobre a notificação

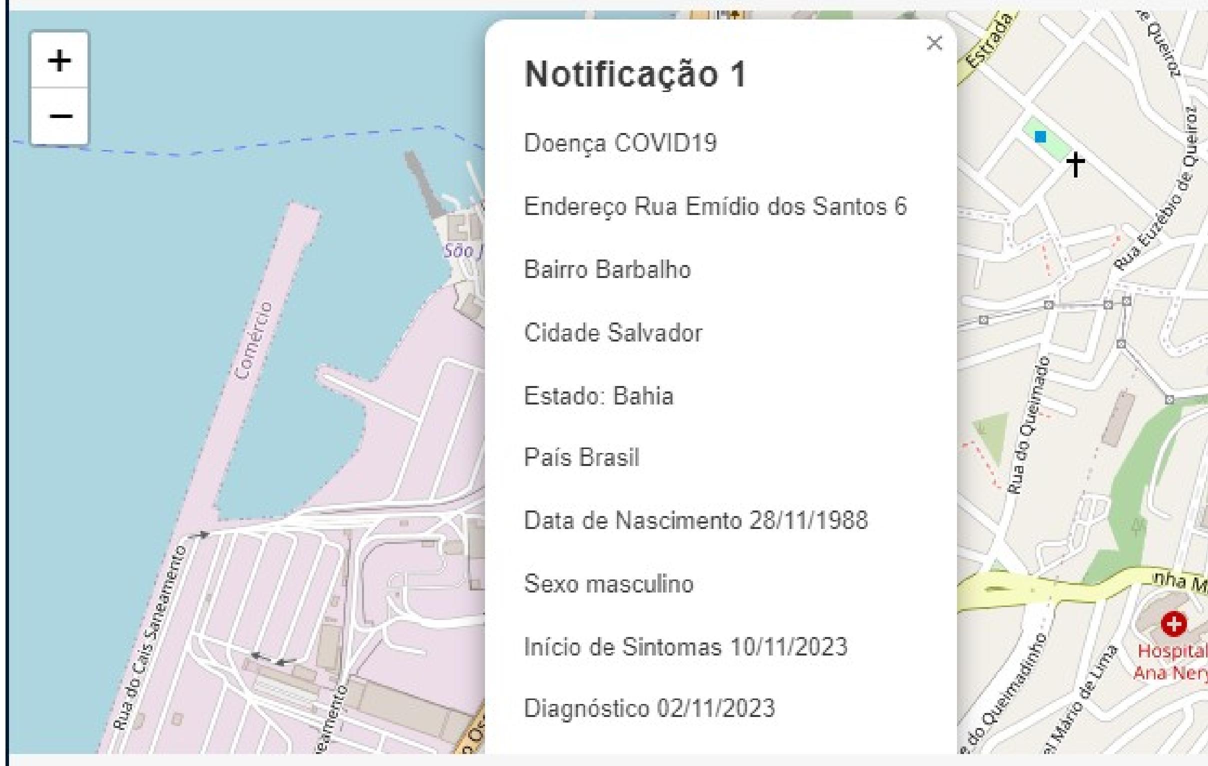

Figura 30: Parte superior das informações das notficações depois de clicar no pino

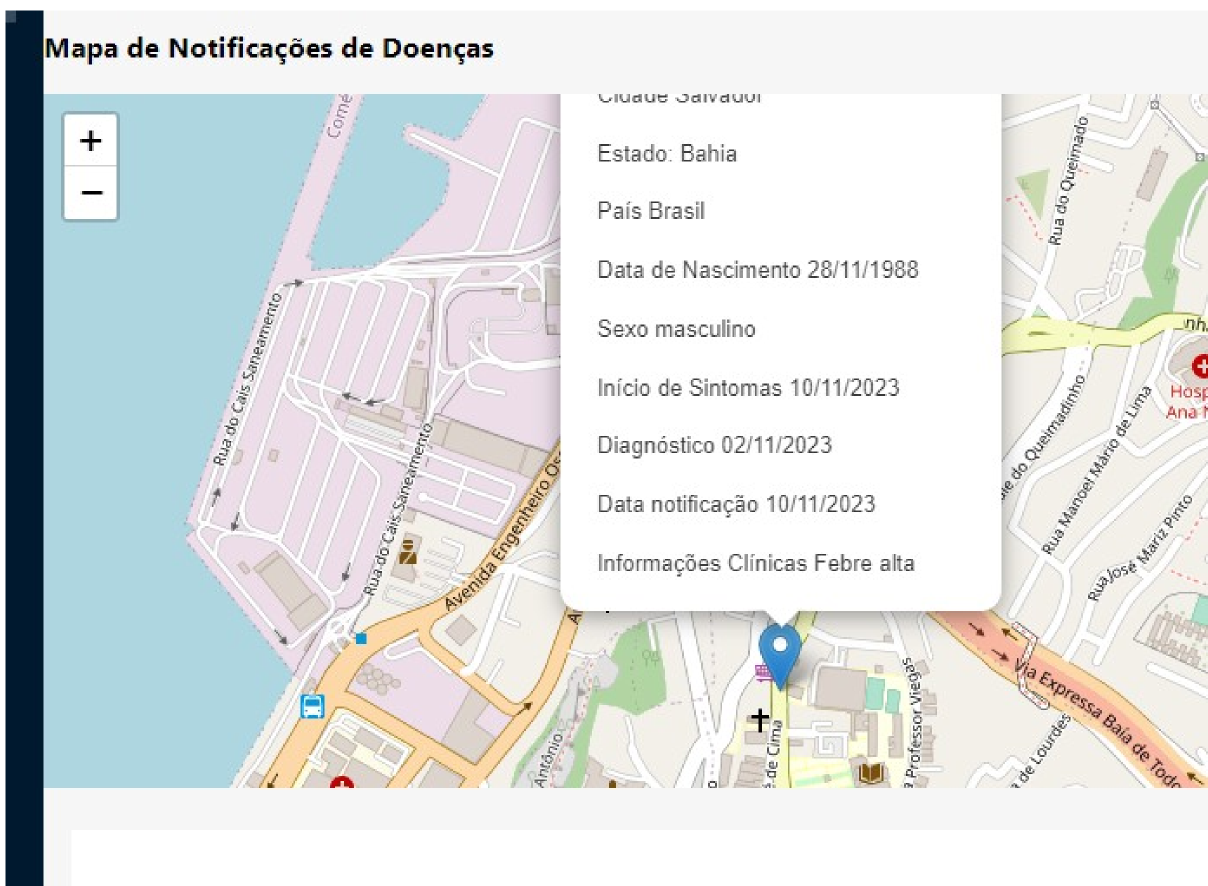

Figura 31: Parte inferior das informações das notifcações informada depois de clicar no pino no mapa dinâmico

E abaixo aparecerá lista de notificações cadastradas

Notificações

id	Doença	Endereço	Bairro	Cidade	Estado	País	Data de Nascimento	Sexo	Data de Inicio de Sintomas	Data de Diagnóstico	Data de Notificação	Informações Clínicas
1	COVID19	Rua Emídio dos Santos 6	Barbalho	Salvador	Bahia	Brasil	28/11/1988	masculino	10/11/2023	02/11/2023	10/11/2023	Febre alta

< 1 >

Figura 32: tabela apresentada abaixo do mapa dinâmico exibindo as notificações cadastradas e suas respectivas informações

Agradecimentos

À Paula Portela, Luísa Brangato, Lúcia Carmem, João Lemos, Daniela Brangato, Letícia Costa, Allan Edgar, Romilson Lopes, Marcelo Diniz, Francine Carvalho, Eduardo Henrique, Wanderlin de Oliveira, George Carvalho, Camila Marinho, Karla Crislaine, Elfo, Orelha, Fofo (em memória), Galileu, Beijú, Aipim, IFBA e ao Colegiado de ADS, Flávia Maristela, Romildo Martins (em memória), Elton Minetto, comunidade Golang e a comunidade Hyperledger, The Linux Foudation, React, React Leaflet, e principalmente ao projeto primoroso Nominatim.

Referências

MINISTÉRIO DA SAÚDE. *Notificação Compulsória*. Disponível em: <https://www.gov.br/saude/pt-br/composicao/svsa/notificacao-compulsoria/notificacao-compulsoria>. Acesso em: 7 out. 2023.

DIÁRIO OFICIAL DA UNIÃO. *Portaria GM/MS Nº 3.328, de 22 de agosto de 2022*. Disponível em: <https://www.gov.br/saude/pt-br/composicao/svsa/notificacao-compulsoria/portaria-gm-ms-no-3-328-de-22-de-agosto-de-2022>. Acesso em: 10 set. 2023.

MINISTÉRIO DA SAÚDE, 2019. *Ficha de Investigação: Acidente de trabalho com exposição à material biológico*. Disponível em: <http://portalsinan.saude.gov.br/images/DRT/DRT_Acidente_Trabalho_Biologico.pdf>. Acesso em: 7 out. 2023.

NAKAMOTO, S. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. Disponível em: https://bitcoin.org/bitcoin.pdf. Acesso em: 20 jun . 2023.

LAMPORT, L.; SHOSTAK, R.; PEASE, M. *The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems (TOPLAS)*, v. 4, n. 3, p. 382-401, 1982.

ONGARO, D., & OUSTERHOUT, J. *In Search of an Understandable Consensus Algorithm*. Disponível em: https://raft.github.io/raft.pdf 2014 . Acesso 20 nov 2023.

LAMPORT,L.*The Part-Time Parliament*. Disponível em: https://lamport.azurewebsites.net/pubs/lamport-paxos.pdf. 1998. Acesso 20 nov 2023

CASTRO, M., & LISKOV, B.Practical Byzantine Fault Tolerance. Disponível em: http://pmg.csail.mit.edu/papers/osdi99.pdf. 1999. Acesso 20 nov 2023

BITCOIN.ORG. *Bitcoin*. Disponível em: https://bitcoin.org/pt_BR. Acesso em: 7 jan. 2023.

ETHEREUM.ORG. *Ethereum*. Disponível em: https://ethereum.org/pt_BR. Acesso em: 7 jan. 2023.

GOLANG.ORG. *Golang*. Disponível em: https://golang.org/ . Acesso em: 21 set. 2023.

REACTJS.ORG. *React.JS*. Disponível em: https://reactjs.org/. Acesso em: 21 set. 2023.

HYPERLEDGER. *Hyperledger – Read The Docs*. Disponível em: <https://hyperledger-fabric.readthedocs.io/>. Acesso em: 10 set. 2023.

PREETHI, Harris. *Blockchain for COVID-19 Patient Health Record*. 2021 5th International Conference on Computing Methodologies and Communication (ICCMC).p. 534-538. 2021. Disponível em https://ieeexplore.ieee.org/document/9418443/. Acesso 10 set 2023

RIMSAN, Mohamed and Mahmood, Ahmad Kamil and Umair, Muhammad and Hassan, Farruk. *COVID-19: A Novel Framework to Globally Track Coronavirus Infected Patients using Blockchain*. p. 70-74. International Conference on Computational Intelligence (ICCI) 2020.

SOUZA, Beatriz Soares de Souza. *PEP + : Um Modelo Blockchain para a Gestão de Casos de Sífilis*. Universidade Federal do Rio Grande do Norte. Departamento de Automação e Computação.2021