Breve introdução da proposta de desafio:

O desafio propõe a criação de um sistema de rastreabilidade para as varinhas mágicas da loja do Sr. Olivaras. Este sistema deve possibilitar que os bruxos na cadeia de produção de varinhas possam listar os materiais que possuem. Assim, o senhor Olivaras, ao utilizar o sistema, pode saber os materiais disponíveis para a produção de uma varinha, bem como adicionar varinhas criadas ao banco de dados. Além disso, por armazenar os dados de forma segura e rastreável utilizando a tecnologia “blockchain”, o sistema servirá como forma de informar os clientes os dados relativos à origem da varinha e dos materiais que a compõem.

1) Abordagem adotada como solução do problema proposto

1.1) Apresentação da estrutura de organização dos dados no World State

Para o armazenamento dos dados dos Materials no World State do Ledger, foram criadas **2 Structs: Wand e Material**. Além disso, o World State contém **2 listas de Type~ID key**: materialIndexList e wandsIndexList.

a) A struct Material possui os seguintes atributos:

```
type Material struct {
   ObjectType string `json:"docType"`
   ID         string `json:"ID"`      // ID único
   Type       string `json:"type"`    // Tipo do Material
   Supplier   string `json:"supplier"`// Supplier
}
```

Todo material será identificado no World State por seu ID. O ID deve ser único para cada material criado. Além do ID, a estrutura Material guarda o“Type” (tipo) do material, de forma que o Sr. Olivaras pode identificar quais os tipos de materiais disponíveis em seu depósito para a construção das varinhas. O Material também contém o atributo “Supplier”, identificando o bruxo da cadeia de produção que forneceu aquele material para o depósito do Sr. Olivaras. É esse atributo que possibilita o rastreamento do fornecedor de origem de determinado material.

- Observações a respeito do ID dos materials:
  O ID deve ser único durante toda a vida do material no sistema. Isso significa que, mesmo que o material deixe de fazer parte da lista de materiais disponíveis ao passar a fazer parte da lista de materiais de uma wand, aquele ID não poderá ser utilizado por outro material cadastrado. O iD só ficará disponível para uso em novo cadastramento caso o material primeiro seja deletado do Worls State pela função “deleteMaterial” ou, nos casos em que ele está na lista de uma wand, a wand a qual esse material pertence seja deletada do sistema. Isso é feito para que todos os materiais, tanto os disponíveis para construção de wands quanto os presentes nas wands possam permanecer no World State e terem sua origem rastreada.

b) A struct Wand possui os seguintes atributos:

```
type Wand struct {
   ObjectType string   `json:"docType"`
   ID         string   `json:"ID"`   // ID único
   Type       string   `json:"type"` // Tipo da wand
   Color      string   `json:"color"`// Cor da wand
   Size       int      `json:"size"` // Tamanho da wand
   Materials  []string `json:"Materials"`// Lista dos IDs dos materiais
usados para a construção da wand}
```

Toda wand será identificada no World State por seu ID. O ID deve ser único para cada wand criada. Além do ID, a estrutura Wand também guarda o “Type” (tipo), de forma que o Sr. Olivaras pode identificar todos os tipos de varinhas prontas em seu estoque. A Wand também contém dados adicionais (e de obrigatório preenchimento) sobre a wand. São eles: “Color” e “Size”. Por fim, a estrutura possui uma lista chamada “Materials ". Essa lista contém todos os IDs dos materials usados naquela varinha. Com isso, o Sr Olivaras poderá consultar para qualquer cliente os dados de origem (Supplier) de qualquer material utilizado na fabricação daquele objeto.

c) Listas do World State:

Para fins de melhor organização e consulta dos materials e wands do sistema, o World State contém 2 listas de Type~ID key: `materialIndexList e wandsIndexList.` Essas listas guardam todos os pares de chaves de Type + ID de todos os materiais disponíveis para a produção de varinhas e todas as varinhas presentes no sistema.

1.2) Funções disponíveis para a criação, consulta e manipulação dos Materials e das Wands

a) Funções básicas requeridas pelo desafio:

- **"initMaterial"** : Cria material no sistema.
  Sequência dos argumentos: “ID”,”Type”,”Supplier”
- **“readMaterial”** : Retorna dos dados do material referente o ID fornecido.
  Sequência dos argumentos: “ID”
- **“getAllMaterials”**: Retorna todos os materiais disponíveis para a fabricação de uma varinha. Não têm argumentos na chamada de função
- **“getTotalNumberOfMaterials”:** Retorna o número total de materiais disponíveis para a fabricação de uma wand. Não têm argumentos na chamada de função
- **“deleteMaterial”:** Deleta do World State todos os dados do material referente ao ID fornecido
  Sequência dos argumentos: “ID”
- **"initWand"** : Cria varinha no sistema.
  Sequência dos argumentos: “ID”,”Type”,”Color”,”Size”,”número de materiais”,”material 1”, “material 2” …
- **“readWand”** : Retorna dos dados da wand referente o ID fornecido.
  Sequência dos argumentos: “ID”

- **“getAllWands”**: Retorna todos as varinhas presentes no sistema. Não têm argumentos na chamada de função.
- **“getTotalNumberOfWands”**: retorna o número total de wands do World State. Não têm argumentos na chamada de função.
- **“deleteWand”**: Deleta do World State todos os dados da wand referente ao ID fornecido.
  Sequência dos argumentos: “ID”

b) Funções adicionais.

Para que o programa ficasse mais completo, foram implementadas algumas funcionalidades além das requisitadas como básicas na descrição do desafio:

- **“getMaterialIndexList”:** Retorna a lista de chave composta do tipo “Type~ID” dos materiais disponíveis para fabricação das wands. Não têm argumentos na chamada de função.
- **“getMaterialsbyType”**: Retorna todos os materiais de determinado “type” disponíveis para fabricação da wands.
  Sequência dos argumentos: “Type”
- “**getNumberMaterialsByType”**: Retorna o número de materiais de determinado “type” disponíveis para fabricação da wands.
  Sequência dos argumentos: “Type”
- **“getAllMaterialsAndIndexList”**: Retorna os dados de todos os materiais disponíveis para fabricação da wands, além da lista de IndexKey desses materiais. Função útil para fins de depuração e revisão do funcionamento do sistema.Não têm argumentos na chamada de função.
- **“getWandsByType”**: Retorna todos as Wands do sistema de determinado “type”.
  Sequência dos argumentos: “Type”
- **“getNumberWandsByType”**: Retorna o número de wands cadastradas de determinado “type”.
  Sequência dos argumentos: “Type”

2) Instruções para execução do sistema

2.1) Inicialização do minifabric:

Para a inicializaçao da ferramenta, é necessário possuir “Docker” Instalado. Após o download da ferramenta “minifabric”, execute o seguinte comando para inicializar a rede padrão:

- sudo ./minifab up

2.2) Instalação do Chaincode Studio no minifab:

Primeiramente, execute o seguinte comando para criar o diretório para o chaincode nos arquivos da ferramenta;

- sudo mkdir -p $(pwd)/vars/chaincode/Studio/go

Depois, manualmente, por linha de terminal, mova todos os arquivos da pasta “Studio” enviada em conjunto com este relatório para a seguinte pasta dentro dos arquivos do minifabric:
$(pwd)/vars/chaincode/Studio/go
Observação: Essa movimentação deve ser feita com “sudo” para ser autorizada.

Por fim, para instalar a chaincode na rede, execute o comando:

- sudo ./minifab ccup -n Studio -l go -v 1.0

Caso dê erro, experimente executar o comando:

- sudo ./minifab ccup -v 2.0 -n Studio -l go

Pronto, com isso, a chaincode já deverá estar instalada e pronta. Os passos acima foram retirados todos da documentação do minifabric.

2.3) Execução das funções de inserção, deleção, manipulação e consulta de Wands e Materials:

Seguindo a documentação oficial do minifabric, disponível no link:
https://github.com/hyperledger-labs/minifabric/blob/main/docs/README.md#invoke-chaincode-methods

Todos as funções descritas neste relatório podem rodados com a execução do seguinte comando:

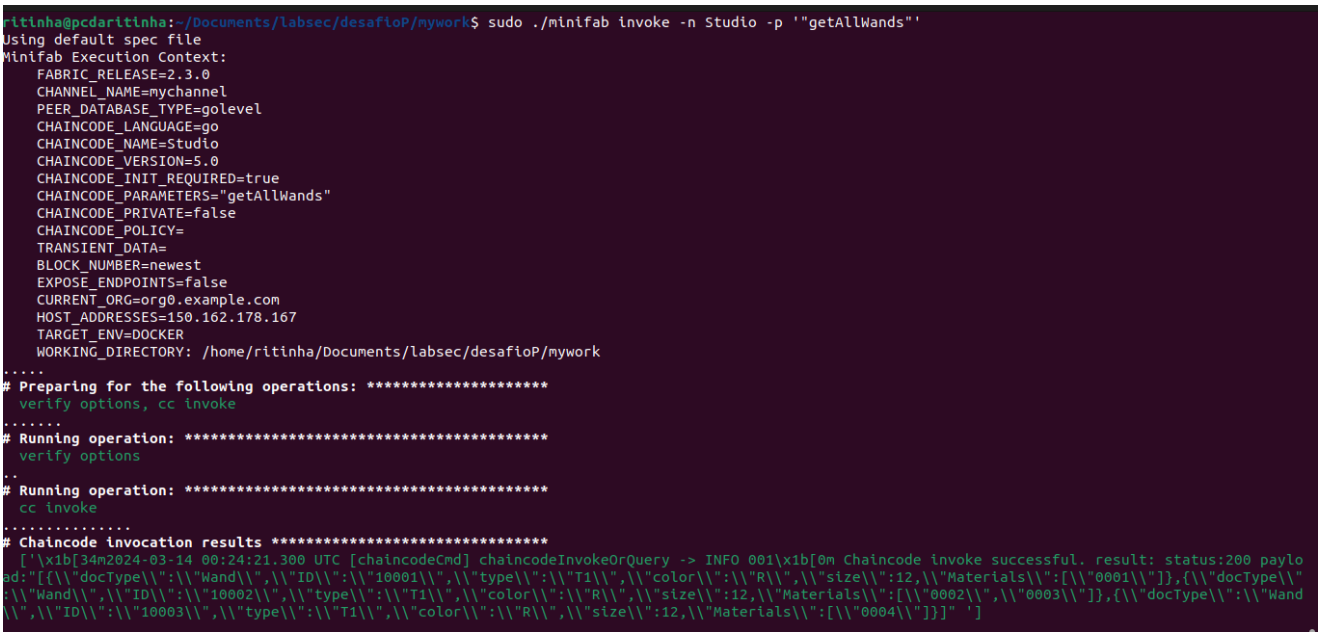
- sudo ./minifab invoke -n Studio -p '"methodname","p1","p2",...'

Substitua o campo ‘methodname' pelo nome da função a ser executada. Além disso, com base na “Sequência dos argumentos” descritos em cada função da seção 1.2 deste relatório, substitua os campos ‘px’ pelo argumento necessário. (Mantenha as aspas duplas e as vírgulas de separação dos argumentos).

3) Exemplo de execução com imagens:

initMaterial: Note que após a criação foi mostrada a lista atual de materiais disponíveis

getAllWands: Note que **os dados são mostrados, por padrão, em formato Json.** Entretanto, é possível observar os dados de cada wand disponível no WorldState. No print, mostra-se os dados de 3 varinhas com ID “10001”, “10002” e “10003” .



4) Fontes de consulta para resolução do desafio

Para o entendimento teórico geral do funcionamento de uma rede Hyperledger Fabric, foi feito o estudo dos capítulos do campo “Key Cooncepts” do site “https://hyperledger-fabric.readthedocs.io”
Também foram assistidas as vídeo aulas de demonstração do funcionamento do minifabric no canal da fundação Hyperledger no youtube.

Para a implementação do código chaincode, foram consultadas 3 fontes principais:

- Tese de graduação “Supply Chain Using Hyperledger Fabric” de Maria-Isavella G. Manolaki-Sempagios.
- Artigo “Writing Chaincode in Golang - the OOP way” de Vishal
- Código “privatemarbles” presente nos arquivos da própria ferramenta Minifabric