INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Projeto MedVault

Equipe

- Líder

  Jorge Abrão Neto

- Membros

  David Samuel Tavares de Moraes

  Lucas Mendes da Silva

  Paulo Victor Rocha de Almeida

Objetivo

Objetivo inicial

Nosso projeto tinha como objetivo inicial, estudar e desenvolver um proxy de re-encriptação para a distribuição de documentos médicos ligados ao armazenamento off chain acoplado a um sistema blockchain.

Objetivos

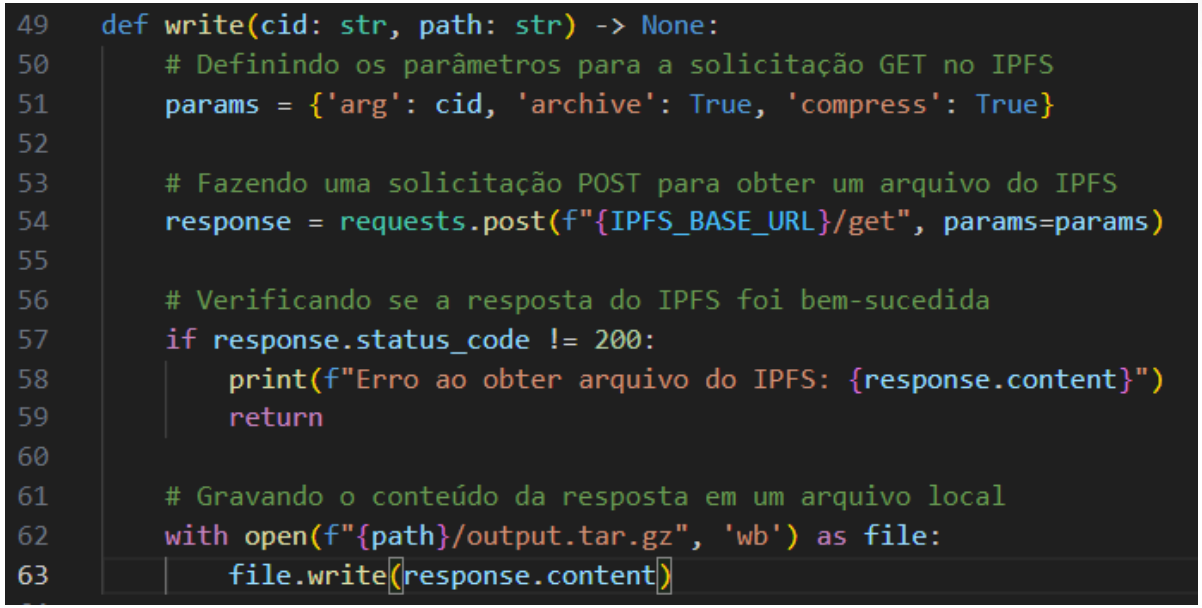
- Desenvolver uma aplicação para armazenamento seguro e descentralizado de registros médicos usando Blockchain e IPFS.
- Desenvolvimento do proxy de r encriptação.
- Desenvolvimento de contratos inteligentes para blockchain.
- Integrar proxy à blockchain.

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Estrutura

IPFS

Solução escolhida para o armazenamento distribuído de dados. O IPFS é uma rede peer-to-peer que utiliza hashes criptográficos para endereçar conteúdo, permitindo que os dados sejam armazenados de forma descentralizada, recuperados eficientemente e resistam à censura. Essa arquitetura distribuída e resistente à censura não apenas melhora a velocidade e a disponibilidade do acesso aos dados, mas também reduz a carga sobre os pontos centrais, superando os desafios tradicionais de escalabilidade e centralização de armazenamento de dados.



```python
def write(cid: str, path: str) -> None:
    # Definindo os parâmetros para a solicitação GET no IPFS
    params = {'arg': cid, 'archive': True, 'compress': True}

    # Fazendo uma solicitação POST para obter um arquivo do IPFS
    response = requests.post(f"{IPFS_BASE_URL}/get", params=params)

    # Verificando se a resposta do IPFS foi bem-sucedida
    if response.status_code != 200:
        print(f"Erro ao obter arquivo do IPFS: {response.content}")
        return

    # Gravando o conteúdo da resposta em um arquivo local
    with open(f"{path}/output.tar.gz", 'wb') as file:
        file.write(response.content)
```

Elementos de robustez presentes

- ## Tolerância

  O Hyperledger é projetado para ser tolerante a falhas, tanto de nós quanto de redes. Os nós do Sawtooth possuem mecanismos de detecção de falhas e são capazes de se recuperar e se reconfigurar em caso de falhas. Além disso, ele possui recursos para lidar com nós maliciosos ou desonestos, garantindo a segurança e a confiabilidade do sistema.

- ## Coordenação

  A coordenação em um sistema Sawtooth é facilitada pelo mecanismo de validação de transações distribuídas. **Cada transação é submetida para validação em diferentes nós, e um algoritmo de consenso é usado para garantir que os nós cheguem a um acordo sobre a validade e a ordem das transações**. Isso requer coordenação eficiente entre nós para sincronizar e validar as transações de forma consistente.

Elementos de robustez presentes

- **Consenso**

O Sawtooth utiliza um mecanismo de consenso modular que significa que diferentes algoritmos de consenso podem ser implementados e escolhidos de acordo com as necessidades do sistema. O algoritmo de consenso Proof of Elapsed Time (PoET) foi implementado através do Sawtooth para, juntamente com alguns mecanismos de validação alcançar um consenso confiável e resistente a falhas.

- **Consistência**

A consistência dos dados é garantida no Sawtooth por meio de um modelo de transações em estado de finalização (Transaction Family State). Cada transação atualiza o estado do blockchain de maneira atômica, garantindo que as operações ocorram de forma consistente e correta. Além disso, o Proof Of Elapsed Time trabalha juntamente com com alguns mecanismos de validação, alcançar um consenso confiável e resistente a falhas.

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Contratos Inteligentes

Smart Contracts:

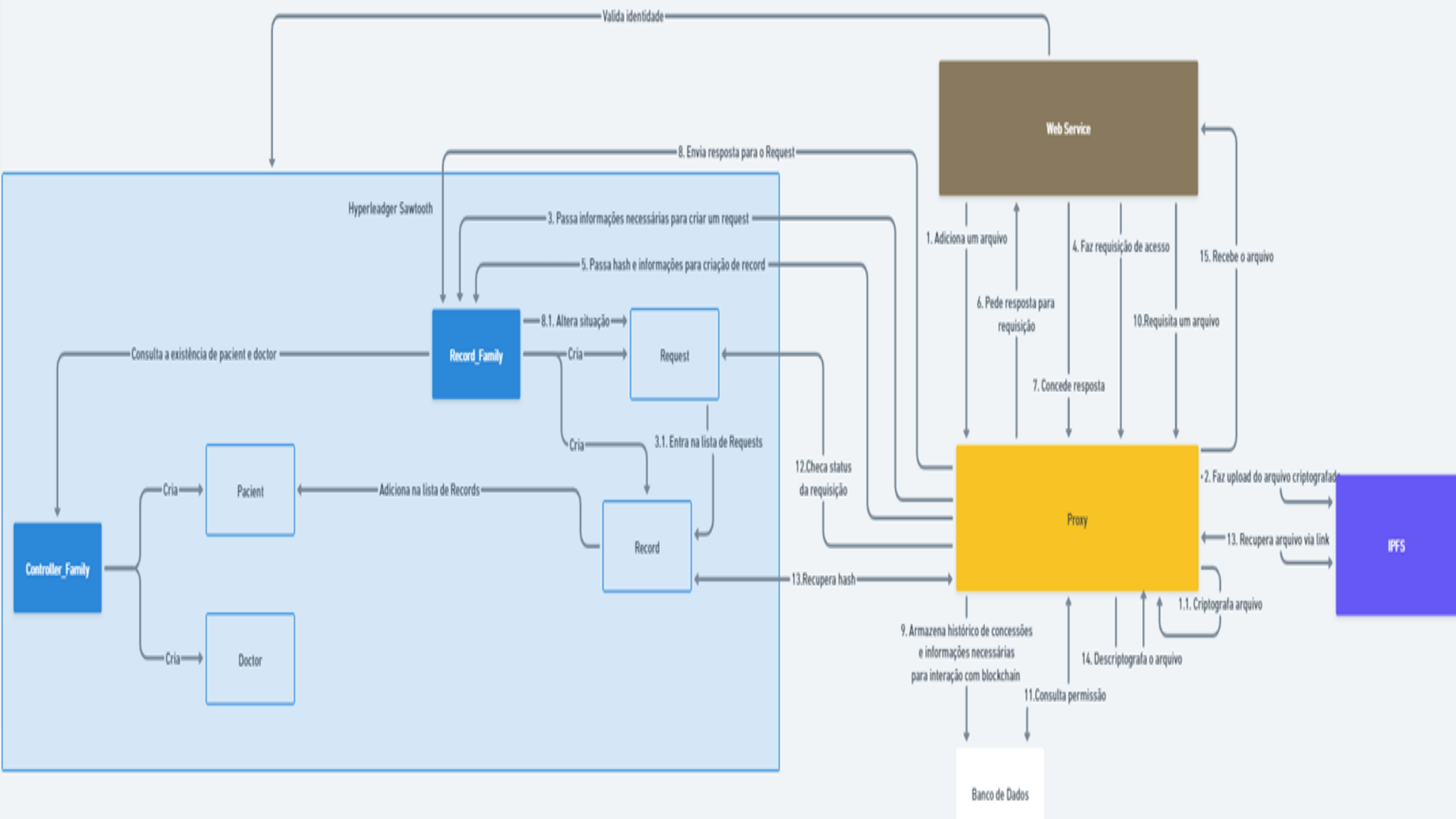
- **O que são?**

Programas computacionais que executam automaticamente as regras acordadas previamente entre as partes. Sua lógica de funcionamento é a mesma de um documento legal tradicional. Ou seja, são estabelecidas obrigações, penalidades, benefícios e outras consequências devidas entre as pessoas que celebram o acordo.

- Agilidade: os contratos inteligentes são baseados em códigos de software,Precisão: sem preenchimentos manuais, as informações se tornam mais precisas. Além disso, elas são atualizadas automaticamente conforme o andamento das transações;
- Autonomia: por falar na ausência de terceiros e na automatização, isso elimina riscos e possíveis erros oriundos da manipulação por outros indivíduos;
- Segurança: os contratos inteligentes são criptografados e distribuídos pelos nós da blockchain. Isso elimina qualquer chance de perda ou alteração não autorizada;
- Confiabilidade: graças à autonomia e segurança dos smart contracts, suas operações são todas rastreáveis e há garantia de que as informações são legítimas.

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Diagrama


Valida identidade
Web Service
8. Envia resposta para o Request
Hyperleadger Sawtooth
3. Passa informações necessárias para criar um request
5. Passa hash e informações para criação de record
1. Adiciona um arquivo
4. Faz requisição de acesso
15. Recebe o arquivo
6. Pede resposta para requisição
10.Requisita um arquivo
7. Concede resposta
Record_Family
8.1. Altera situação
Consulta a existência de pacient e doctor
Cria
Request
Cria
3.1. Entra na lista de Requests
12.Checa status da requisição
Cria
Pacient
Adiciona na lista de Records
Proxy
2. Faz upload do arquivo criptografado
IPFS
Controller_Family
Record
13. Recupera arquivo via link
13.Recupera hash
1.1. Criptografa arquivo
Cria
Doctor
9. Armazena histórico de concessões e informações necessárias para interação com blockchain
14. Descriptografa o arquivo
11.Consulta permissão
Banco de Dados

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Family_Controller

```python
from sawtooth_sdk.processor.handler import TransactionHandler

from models.doctor_model import Doctor
from models.patient_model import Patient
from utils import _hash

import json

FAMILY_NAME = 'FAMILY_CONTROLLER'

class ControllerTransactionHandler(TransactionHandler):
    def __init__(self):
        self._namespace_prefix = _hash(FAMILY_NAME.encode('utf-8'))[0:6]
    @property
    def family_name(self):
        return FAMILY_NAME

    @property
    def family_versions(self):
        return ['1.0']

    @property
    def namespaces(self):
        return [self._namespace_prefix]

    def getNamespace(self):
        return self._namespace_prefix

    def apply(self, transaction, context):
        try:
            header = transaction.header
            signer = header.signer_public_key
            action, payload = ControllerFactory.from_bytes(transaction.payload)
            address = self._namespace_prefix + _hash(payload.cpf.encode('utf-8'))[:64]
            state = context.get_state([address])
        except:
            print("Um erro ocorreu")
            return
```

```
40          if action == 'add':
41              if state:
42                  print(f'{payload._type} already exists')
43                  return None
44              state_data = payload.to_bytes()
45              context.set_state({address: state_data})
46              print(f'{payload._type} was added')
47
48          elif action == 'show':
49              if not state:
50                  print(f'{payload._type} does not exist')
51                  return None
52              state_data = state[0].data.decode('utf-8')
53              print(f'{payload._type} data: {ControllerFactory.getPatient(state)}')
54
55          elif action == 'delete':
56              if not state:
57                  print('{} does not exist' %format(payload._type))
58                  return None
59              context.delete_state([address])
60              print(f'{payload._type} was deleted')
61
```

```python
class ControllerFactory:
    @staticmethod
    def getPatient(state):
        if not state:
            print('Patient does not exist')
            return None
        state_data = state[0].data
        return Patient.from_bytes(state_data)
    @staticmethod
    def getPayload(payload):
        try:
            # The payload is csv utf-8 encoded string
            data = json.loads(payload.decode())
            action = data["action"]
            type = data["type"]
            body = data["body"]
        except ValueError:
            print("Invalid payload serialization")
            return None

        if not action:
            print('Action is required')
            return None

        if action not in ('add', 'show', 'delete'):
            print('Invalid action: {}' % format(action))
            return None

        if type not in ('doctor','patient'):
            print('Invalid type: {}'%format(type))
            return None

        if type == 'doctor':
            return action, Doctor(body)

        if type == 'patient':
            return action, Patient(body)
    @staticmethod
    def from_bytes(payload):
        return ControllerFactory.getPayload(payload=payload)
```

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Record_Family

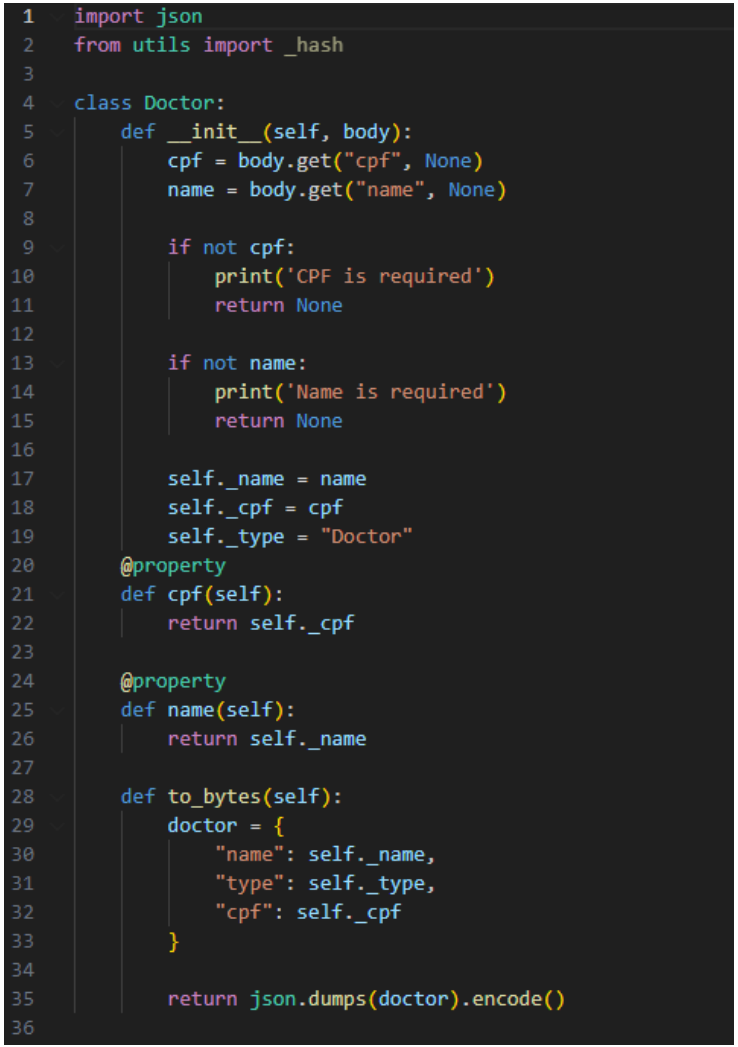
```python
from sawtooth_sdk.processor.handler import TransactionHandler

from models.record_model import Record
from utils import _hash
from families.controller_family import ControllerTransactionHandler, ControllerFactory

import json

FAMILY_NAME = 'FAMILY_RECORD'

class RecordTransactionHandler(TransactionHandler):
    def __init__(self):
        self._namespace_prefix = _hash(FAMILY_NAME.encode('utf-8'))[0:6]

    @property
    def family_name(self):
        return FAMILY_NAME

    @property
    def family_versions(self):
        return ['1.0']

    @property
    def namespaces(self):
        return [self._namespace_prefix]

    def apply(self, transaction, context):
        header = transaction.header
        signer = header.signer_public_key
        action, patient_cpf, body = RecordFactory.from_bytes(transaction.payload)

        controller_namespace_prefix = ControllerTransactionHandler().getNamespace()
        patient_address = controller_namespace_prefix + _hash(patient_cpf.encode('utf-8'))[:64]
        state = context.get_state([patient_address])
        patient = ControllerFactory.getPatient(state)
```

```
37          if action == 'add':
38              record = Record(body)
39              patient.add_record(record)
40              print(f"Record added")
41          elif action == 'show':
42              id_record = body["id_record"]
43              record = patient.get_record(id_record)
44              print(f"Record {repr(record)}")
45          elif action == 'delete':
46              id_record = body["id_record"]
47              patient.delete_record(id_record)
48          elif action == 'grant':
49              id_record = body["id_record"]
50              doctor_cpf = body["doctor_cpf"]
51              id_request = body["id_request"]
52              request_status = body["request_status"]
53              patient.grant_record(id_record, doctor_cpf, id_request, request_status)
54              print(f"Record granted")
55              record = patient.get_record(id_record)
56              print(f"Record {repr(record)}")
57          elif action == 'request':
58              id_record = body["id_record"]
59              doctor_cpf = body["doctor_cpf"]
60              id_request = body["id_request"]
61              patient.request_record(id_record, doctor_cpf, id_request)
62              print(f"Record requested")
63          patient.update_patient(state=state, address=patient_address, context=context)
64          print(patient)
```

```python
class RecordFactory:
    @staticmethod
    def getPayload(payload):
        try:
            # The payload is csv utf-8 encoded string
            data = json.loads(payload.decode())
            action = data["action"]
            patient_cpf = data["patient_cpf"]
            body = data["body"]

        except ValueError:
            print("Invalid payload serialization")
            return None

        if not action:
            print('Action is required')
            return None

        if action not in ('add', 'show', 'delete', 'grant', 'request'):
            print(f'Invalid action: {action}')
            return None
        return action, patient_cpf, body
    @staticmethod
    def from_bytes(payload):
        return RecordFactory.getPayload(payload=payload)
```

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Entidades



```python
import json
from utils import _hash

class Doctor:
    def __init__(self, body):
        cpf = body.get("cpf", None)
        name = body.get("name", None)

        if not cpf:
            print('CPF is required')
            return None

        if not name:
            print('Name is required')
            return None

        self._name = name
        self._cpf = cpf
        self._type = "Doctor"
    @property
    def cpf(self):
        return self._cpf

    @property
    def name(self):
        return self._name

    def to_bytes(self):
        doctor = {
            "name": self._name,
            "type": self._type,
            "cpf": self._cpf
        }

        return json.dumps(doctor).encode()

```

```python
import json
from utils import _hash

from models.record_model import Record

class Patient:
    def __init__(self, body):
        cpf = body.get("cpf", None)
        name = body.get("name", None)
        records = body.get("records", None)

        if not cpf:
            print('CPF is required')
            return None

        if not name:
            print('Name is required')
            return None

        if records:
            records = [Record.from_json(record) for record in records]

        self._name = name
        self._cpf = cpf
        self._records = records or []
        self._type = "Patient"

    @property
    def cpf(self):
        return self._cpf

    @property
    def name(self):
        return self._name

    @staticmethod
    def from_bytes(data):
        body = json.loads(data.decode('utf-8'))
        return Patient(body)

    def to_bytes(self):
        patient = {
            "name": self._name,
            "cpf": self._cpf,
            "type": self._type,
            "records": [record.to_json() for record in self._records]
        }
        return json.dumps(patient).encode()
```

```python
    def update_patient(self, state, address, context):
        state_data = self.to_bytes()
        context.set_state({address: state_data})

    def add_record(self, record):
        self._records.append(record)

    def get_record(self, id_record):
        for record in self._records:
            if record._id_record == id_record:
                return record
        print('Record does not exist')
        return None

    def delete_record(self, id_record):
        for record in self._records:
            if record._id_record == id_record:
                self._records.remove(record)
                return None
        print('Record does not exist')
        return None


    def request_record(self, id_record, doctor_cpf, id_request):
        for record in self._records:
            if record._id_record == id_record:
                record.requested(id_request, doctor_cpf)
                return None
        print('Record does not exist')
        return None

    def grant_record(self, id_record, doctor_cpf, id_request, request_status):
        for record in self._records:
            if record._id_record == id_record:
                record.granted(id_request, request_status)
                return None
        print('Record does not exist')
        return None

    def __repr__(self) -> str:
        return f"Name: {self._name}, CPF: {self._cpf}, Type: {self._type}, Records: {';'.join([repr(record) for record in self._records])}\n"
```

```python
import json
from utils import _hash
from models.request_model import Request
class Record:
    def __init__(self, body):
        id_record = body.get("id_record",None)
        title = body.get("title", None)
        bundle_hash = body.get("bundle_hash", None)
        requests = body.get("requests", None)

        if not id_record:
            print('ID is required')
            return None

        if requests:
            requests = [Request.from_json(request) for request in requests]

        self._id_record = id_record
        self._title = title
        self._bundle_hash = bundle_hash
        self._requests = requests or []
    @property
    def action(self):
        return self._action

    @property
    def title(self):
        return self._title

    @property
    def bundle_hash(self):
        return self._bundle_hash

    def to_json(self):
        record = {
            "id_record": self._id_record,
            "title": self._title,
            "bundle_hash": self._bundle_hash,
            "requests": [request.to_json() for request in self._requests]
        }

        return json.dumps(record)
```

```
    @staticmethod
    def from_json(record):
        return Record(json.loads(record))

    def requested(self, id_request, doctor_cpf):

        request = Request({
            "request_id": id_request, "doctor_cpf": doctor_cpf
        })
        self._requests.append(request)

    def granted(self, id_request, request_status):
        for request in self._requests:
            if request._request_id == id_request:
                request.reply(request_status)
                return None
        print('Record does not exist')
        return None
    def __repr__(self):
        return f"ID: {self._id_record}, bundle_hash: {self._bundle_hash}, title: {self._title}, Requests: --- {','.join([repr(request) for request in self._requests])} ---\n"
```

```python
import json
from utils import _hash

class Request:
    def __init__(self, body):
        doctor_cpf = body.get("doctor_cpf", None)
        request_id = body.get("request_id", None)
        request_status = body.get("request_status", None)

        if request_status == None:
            request_status = 0

        self._doctor_cpf = doctor_cpf
        self._request_id = request_id
        self._request_status = request_status #0 for waiting approval, 1 for approved, 2 for denied.

    @property
    def doctor(self):
        return self._doctor_cpf

    @property
    def id(self):
        return self._request_id

    @property
    def status(self):
        return self._request_status

    def reply(self, status):
        self._request_status = status

    def to_json(self):
        request = {
            "doctor_cpf": self._doctor_cpf,
            "request_id": self._request_id,
            "request_status": self._request_status
        }

        return json.dumps(request)
```

```
42      @staticmethod
43      def from_json(data):
44          data_dict = json.loads(data)
45
46          # Extract the values for doctor_cpf and request_id
47
48          request = Request(data_dict)
49
50          return request
51
52      def __repr__(self):
53          return f"ID: {self._request_id}, Doctor_CPF: {self._doctor_cpf}, request_status: {self._request_status}"
```

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

Re-criptografia

Interface IPFS método add:

```python
def add(file) -> str:
    # Faz uma solicitação POST para adicionar um arquivo ao IPFS
    response = requests.post(f"{IPFS_BASE_URL}/add", files={"file": file}, params={'cid-version': 1})

    # Verifica se a resposta do IPFS foi bem-sucedida
    if response.status_code != 200:
        print(f"Erro ao obter arquivo do IPFS: {response.content}")
        return

    # Analisa os dados JSON retornados pela resposta
    data = json.loads(response.content)

    # Retorna o hash do CID (Content Identifier) do arquivo adicionado ao IPFS
    return data["Hash"]
```

Interface IPFS método get:

```python
def get(cid: str) -> dict:
    # Parâmetros para a solicitação GET no IPFS
    params = {'arg': cid, 'archive': True, 'compress': True}

    # Faz uma solicitação POST para obter um arquivo do IPFS
    response = requests.post(f"{IPFS_BASE_URL}/get", params=params)

    # Verifica se a resposta do IPFS foi bem-sucedida
    if response.status_code != 200:
        print(f"Erro ao obter arquivo do IPFS: {response.content}")
        return

    try:
        # Importa as bibliotecas necessárias para manipulação de arquivos tar e io
        import tarfile, io

        # Abre o arquivo tar recebido como resposta e lê o seu conteúdo
        with tarfile.open(fileobj=io.BytesIO(response.content), mode='r:gz') as tar:
            filename = tar.getnames()[0]
            file = tar.extractfile(filename)
            content: bytes = file.read()

        # Converte o conteúdo lido (bytes) para um dicionário usando JSON
        return json.loads(content)
    except tarfile.BadGzipFile:
        print("Erro ao obter arquivo do IPFS: Não foi possível realizar leitura do arquivo - Arquivo não é Gzip")
    except KeyError:
        print(f"Erro ao obter arquivo do IPFS: Não foi possível extrair arquivo {filename} - Arquivo não existe")
```

Interface IPFS método write:

```python
def write(cid: str, path: str) -> None:
    # Definindo os parâmetros para a solicitação GET no IPFS
    params = {'arg': cid, 'archive': True, 'compress': True}

    # Fazendo uma solicitação POST para obter um arquivo do IPFS
    response = requests.post(f"{IPFS_BASE_URL}/get", params=params)

    # Verificando se a resposta do IPFS foi bem-sucedida
    if response.status_code != 200:
        print(f"Erro ao obter arquivo do IPFS: {response.content}")
        return

    # Gravando o conteúdo da resposta em um arquivo local
    with open(f"{path}/output.tar.gz", 'wb') as file:
        file.write(response.content)
```

Encriptação do token de acesso (simula o trabalho da blockchain):

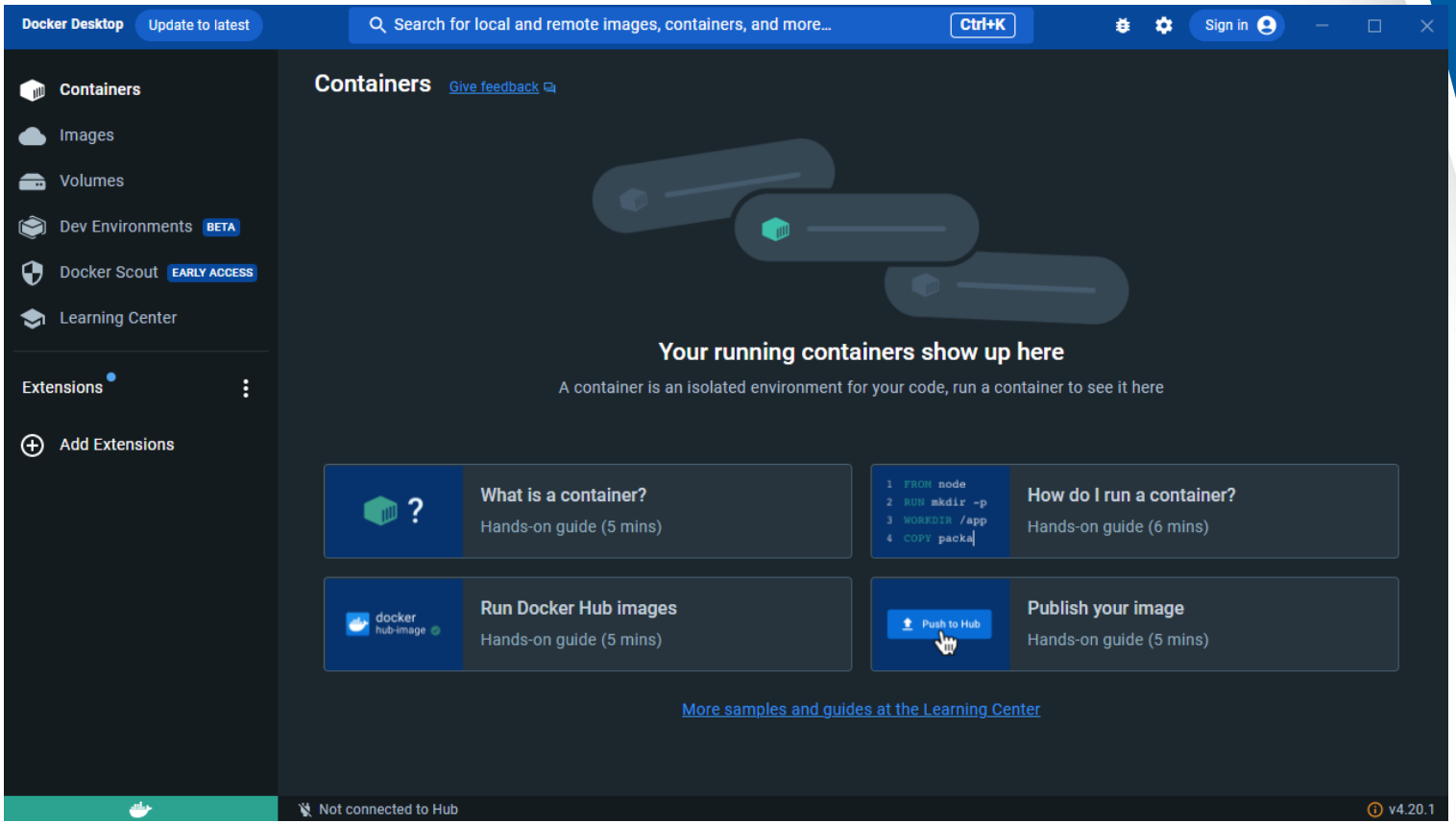
```python
def create_instance(content: bytes) -> None:
    patient_keys = keygen()
    doctor_keys = keygen()
    patient_signer_keys = keygen()
    patient_signer = Signer(patient_signer_keys['secret_key'])
    doctor_id = "1"
    record = "RECORD 1"
    key_frags = patient_grants_access(patient_keys['secret_key'], patient_signer, doctor_keys['public_key'], 1, 1)

    cid_record = patient_create_record(content, patient_keys['public_key'])
    print(f"O Record do paciente foi armazenado na IPFS em {cid_record}")

    data = {
        'doctor_id': doctor_id,
        'public_key_patient': serializeFrom(bytes(patient_keys['public_key'])),
        'public_key_signer_patient': serializeFrom(bytes(patient_signer_keys['public_key'])),
        'public_key_doctor': serializeFrom(bytes(doctor_keys['public_key'])),
        'key_frag':  serializeListFrom(key_frags),
        'link': cid_record
    }
    # Armazenando o pedido do SC em um arquivo
    with open(SC_FILE_STORAGE, 'w') as file:
        json.dump(data, file, indent=4)
    print("Arquivo do Contrato Inteligente foi criado")

    # Criando o armazenamento do Doutor:
    with open(DOCTOR_FILE_STORAGE, 'w') as file:
        keys = serializeKeys(doctor_keys)
        json.dump(keys, file, indent=4)
    print("Arquivo com as chaves do Doutor foi criado")
```

Re-encriptação:

```python
def test_instance():
    # Armazenando o pedido do SC em um arquivo
    with open(SC_FILE_STORAGE, 'r') as file:
        token_obj = json.load(file)
    print("Token obtido do armazenamento do SC")

    # Criando o armazenamento do Doutor:
    with open(DOCTOR_FILE_STORAGE, 'r') as file:
      doctor_key_encoded = json.load(file)
      doctor_keys = deserializeKeys(doctor_key_encoded)
    print("Chaves obtidas do armazenamento do doutor")

    public_key_patient = PublicKey.from_bytes(deserializeFrom(token_obj["public_key_patient"]))
    public_key_signer_patient = PublicKey.from_bytes(deserializeFrom(token_obj["public_key_signer_patient"]))
    public_key_doctor = PublicKey.from_bytes(deserializeFrom(token_obj["public_key_doctor"]))
    key_frag = deserializeListFrom(VerifiedKeyFrag.from_verified_bytes, token_obj["key_frag"])
    print("Token decodificado")


    ipfs_obj = ipfs.get(token_obj["link"])
    capsule = Capsule.from_bytes(deserializeFrom(ipfs_obj['capsule']))
    ciphertext = deserializeFrom(ipfs_obj['content'])
    print("Cifra e capsula obtidas do IPFS")

    cfrags = get_cfrags(public_key_signer_patient, public_key_patient, public_key_doctor, capsule, kfrags=key_frag)
    content: bytes = decrypt_reencrypted(
            verified_cfrags = cfrags,
            receiving_sk = doctor_keys["secret_key"], delegating_pk = public_key_patient,
            capsule = capsule, ciphertext = ciphertext
        )
    print("Conteudo decodificado por reencriptacao")
    return content
```

INSTITUTO DE INFORMÁTICA
Universidade Federal de Goiás

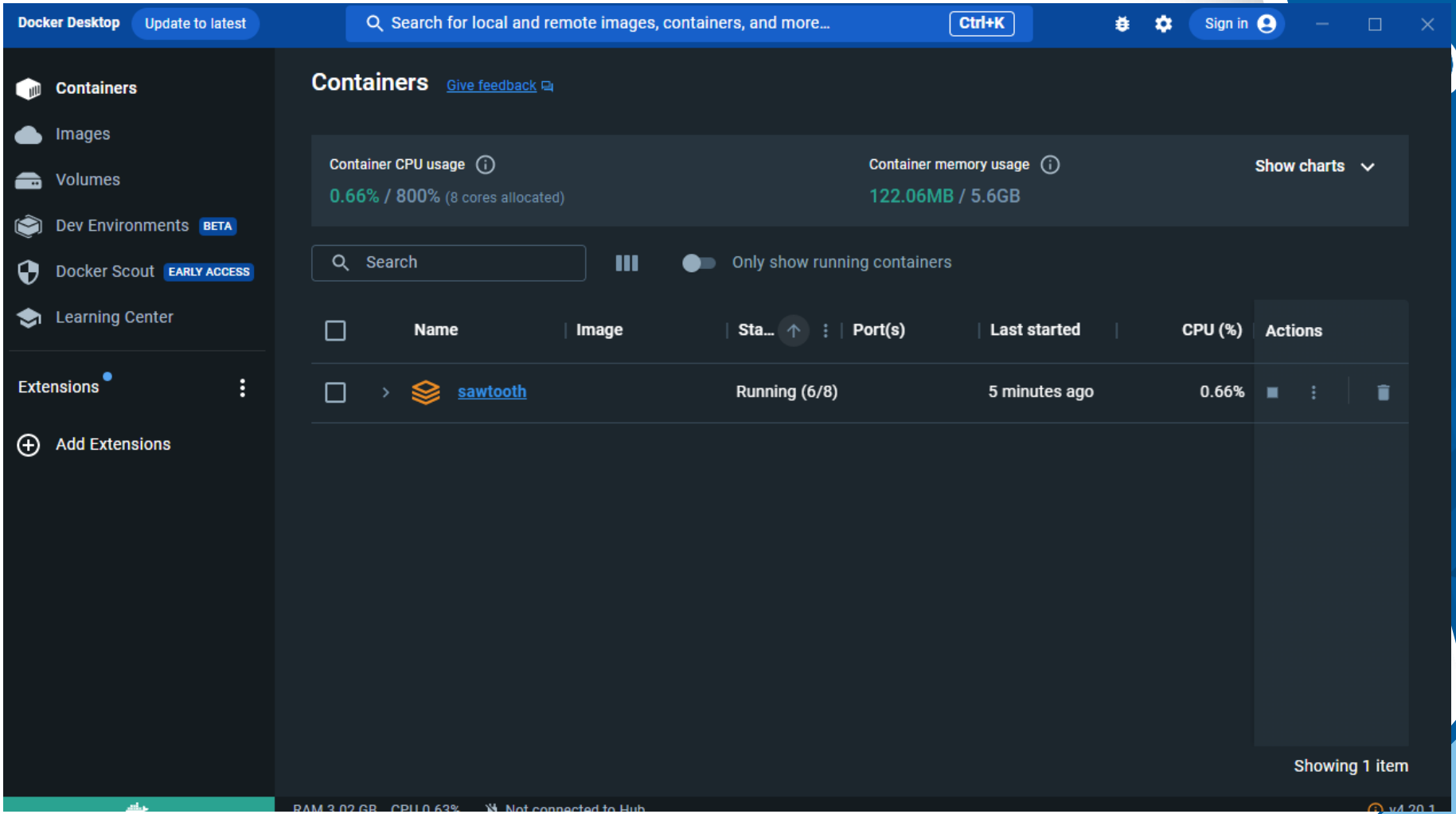
Demonstração



Docker Desktop

```
PS C:\sawtooth-example\sawtooth> docker compose -f sawtooth-default.yaml up -d
[+] Building 0.0s (0/0)
[+] Running 8/8
 ✓ Container sawtooth-validator-default            Started                    0.5s
 ✓ Container sawtooth-devmode-engine-rust-default  Started                    2.1s
 ✓ Container sawtooth-intkey-tp-python-default     Started                    2.0s
 ✓ Container sawtooth-settings-tp-default          Started                    2.1s
 ✓ Container sawtooth-rest-api-default             Started                    1.5s
 ✓ Container sawtooth-xo-tp-python-default         Started                    1.5s
 ✓ Container sawtooth-shell-default                Started                    2.6s
 ✓ Container sawtooth-dev-default                  Started                    2.4s
PS C:\sawtooth-example\sawtooth>
```

```
PS C:\sawtooth-example\sawtooth> docker exec -t sawtooth-dev-default bash -c "cd sawtooth/server/ ; python3 main.py"
Servidor iniciou
```


Docker Desktop
Update to latest
Search for local and remote images, containers, and more...
Ctrl+K
Sign in
Containers
Images
Volumes
Dev Environments
BETA
Docker Scout
EARLY ACCESS
Learning Center
Extensions
Add Extensions
Containers
Give feedback
Container CPU usage
0.66% / 800% (8 cores allocated)
Container memory usage
122.06MB / 5.6GB
Show charts
Search
Only show running containers
Name
Image
Sta...
Port(s)
Last started
CPU (%)
Actions
sawtooth
Running (6/8)
5 minutes ago
0.66%
Showing 1 item
RAM 3.02 GB
CPU 0.63%
Not connected to Hub
v4.20.1

```
rawPayload = {
    "action": "add",
    "type": "doctor",
    "body": {
        "cpf": DOCTOR_CPF,
        "name": "ryan",
    }
}
client = PIBITIClient(baseUrl=DEFAULT_URL, keyFile=key_file, family_name=CONTROLLER_FAMILY_NAME, cpf=DOCTOR_CPF)
print("Wrap and send - Add Doctor")
response = client._wrap_and_send(rawPayload)
print("Response: {}".format(response))
```

```
PS C:\sawtooth-example\sawtooth> docker exec -t sawtooth-dev-default bash -c "cd sawtooth/client/ ; python3 dapp_test.py"
Wrap and send - Add Doctor
Response: {
  "link": "http://rest-api:8008/batch_statuses?id=2befb43d15d6cf44e8004c12be2ece209ed486b8a2ba081f2752a888a7defb08549e75fb49f285a92db79bed251d664156d821d14411666cb0860867106af821"
}
PS C:\sawtooth-example\sawtooth>
```

```
docker exec -t sawtooth-dev-default bash -c "cd sawtooth/server/ ; python3 main.py"
Servidor iniciou
Doctor was added
Doctor was added
```

```
rawPayload = {
    "action": "add",
    "type": "patient",
    "body": {
        "cpf": PATIENT_CPF,
        "name": "Joana",
    }
}
client = PIBITIClient(baseUrl=DEFAULT_URL, keyFile=key_file, family_name=CONTROLLER_FAMILY_NAME, cpf=PATIENT_CPF)
print("Wrap and send - Add Patient")
response = client._wrap_and_send(rawPayload)
print("Response: {}".format(response))
```

```
PS C:\sawtooth-example\sawtooth> docker exec -t sawtooth-dev-default bash -c "cd sawtooth/client/ ; python3 dapp_test.py"
Wrap and send - Add Patient
Response: {
  "link": "http://rest-api:8008/batch_statuses?id=b1047d02d84bc622a571af937ef81d0469239d077c27177856faedd70bfa4fdf4df6fc0f1ed5b94792c129f5bd2f7c89b1272
d8d0823a0454e4282d10d250f24"
}
PS C:\sawtooth-example\sawtooth>
```

```
docker exec -t sawtooth-dev-default bash -c "cd sawtooth/server/ ; python3 main.py"
Servidor iniciou
Doctor was added
Doctor was added
Patient was added
Patient was added
```

```python
rawPayload = {
    "action": "add",
    "patient_cpf": PATIENT_CPF,
    "body": {
        "id_record": "1",
        "title": "Record Title 1",
        "bundle_hash": "BUNDLE_HASH12312312",
    }
}
client = PIBITIClient(baseUrl=DEFAULT_URL, keyFile=key_file, family_name=RECORD_FAMILY_NAME, cpf=PATIENT_CPF)
print("Wrap and send - Add Record")
response = client._wrap_and_send(rawPayload)
print("Response: {}".format(response))
```

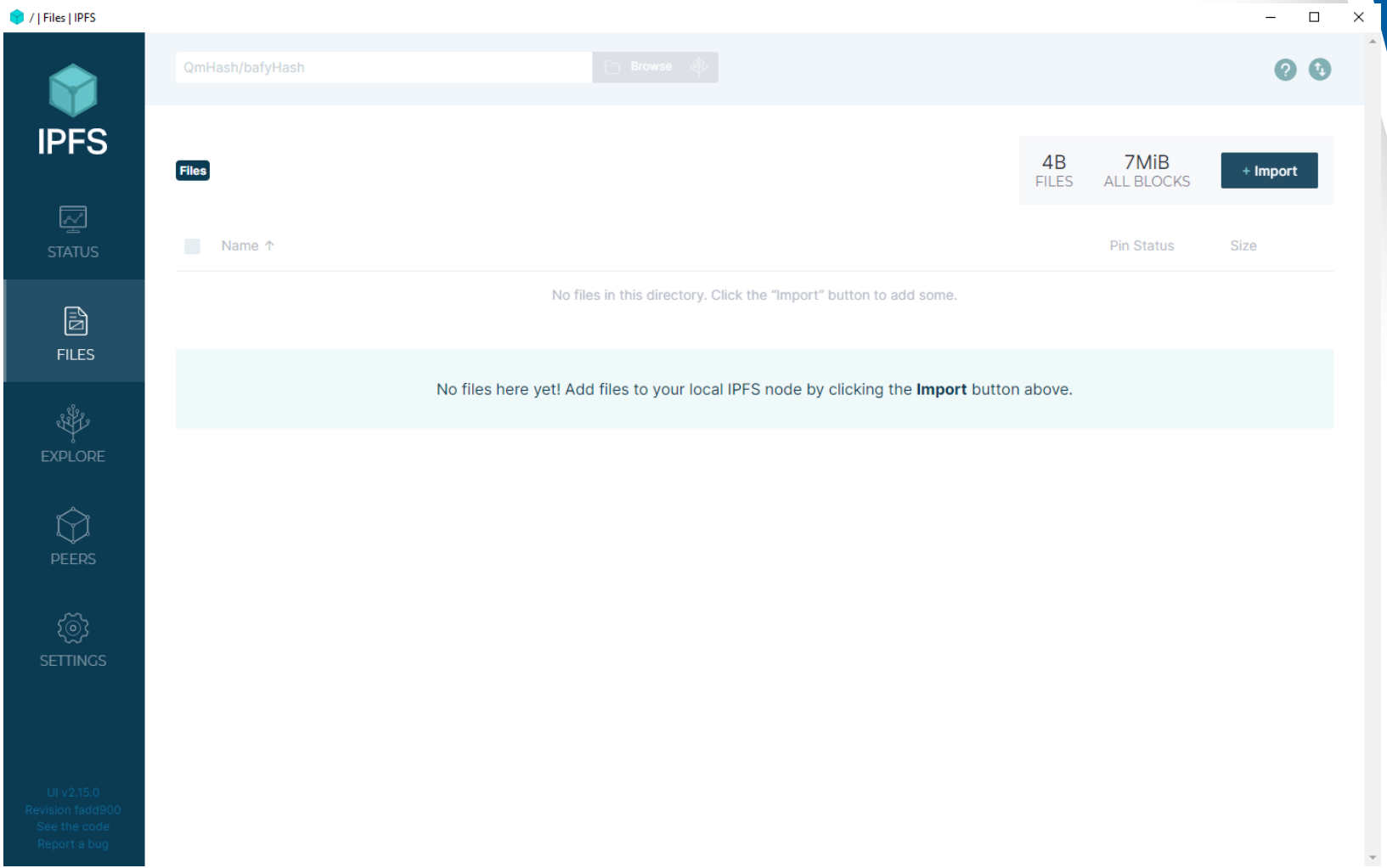
```
PS C:\sawtooth-example\sawtooth> docker exec -t sawtooth-dev-default bash -c "cd sawtooth/client/ ; python3 dapp_test.py"
Wrap and send - Add Record
Response: {
  "link": "http://rest-api:8008/batch_statuses?id=886a9642ba003965caa9944752c0426bece72d8786eda75b1286de77d2d34f4c60f56ba6b63a1d3f7d61910f41a68db45ac93ae629a25ea33c9dcdd90587ebdd"
}
PS C:\sawtooth-example\sawtooth>
```

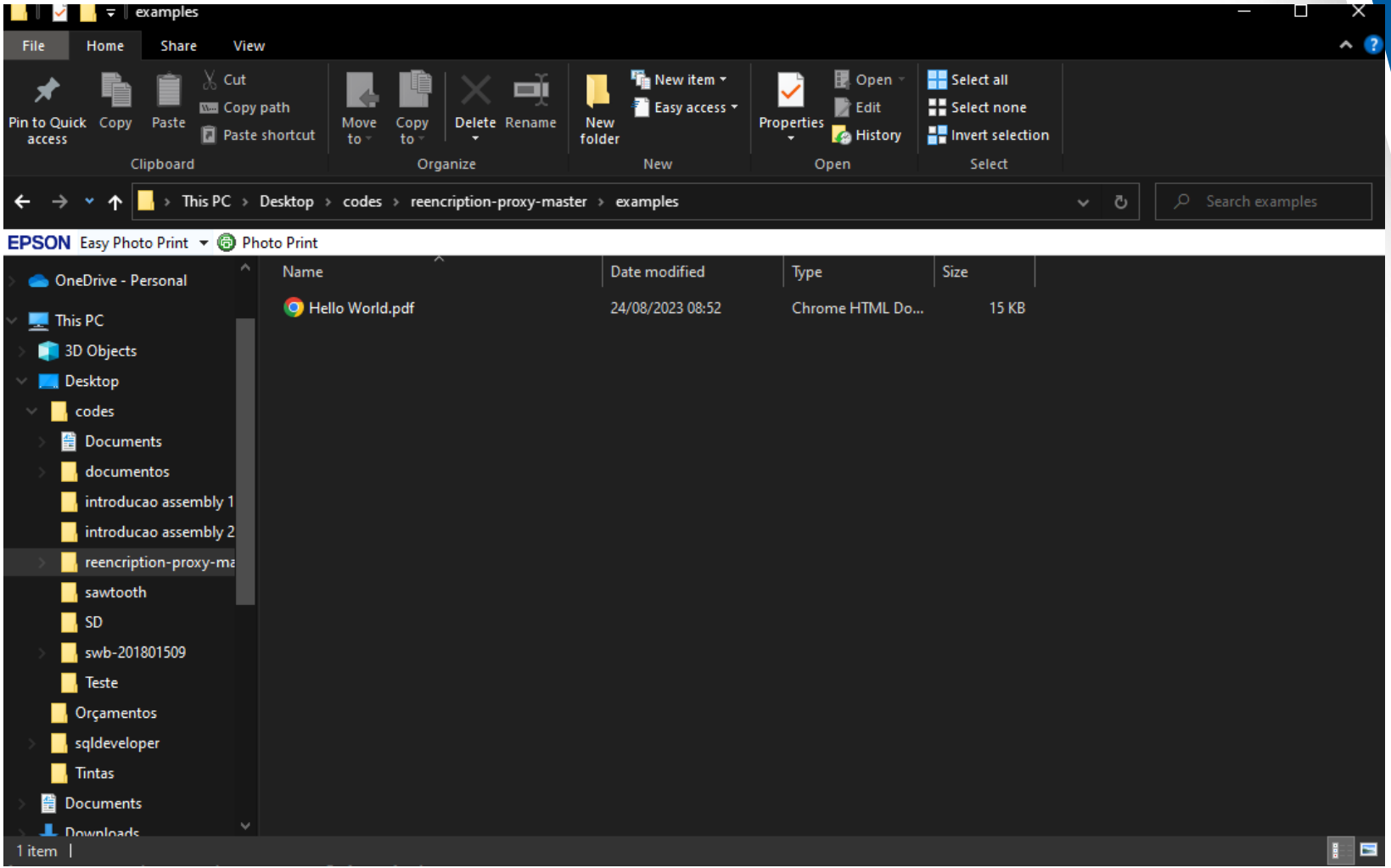
```
Doctor was added
Doctor was added
Patient was added
Patient was added
Record added
Name: Joana, CPF: 222.222.222-22, Type: Patient, Records: ID: 1, bundle_hash: BUNDLE_HASH12312312, title: Record Title 1, Requests: ---  ---


Record added
Name: Joana, CPF: 222.222.222-22, Type: Patient, Records: ID: 1, bundle_hash: BUNDLE_HASH12312312, title: Record Title 1, Requests: ---  ---
```

```
rawPayload = {
    "action": "request",
    "patient_cpf": PATIENT_CPF,
    "body": {
        "id_record": "1",
        "doctor_cpf": DOCTOR_CPF,
        "id_request": "ID_rEQUEST1",
    }
}
client = PIBITIClient(baseUrl=DEFAULT_URL, keyFile=key_file, family_name=RECORD_FAMILY_NAME, cpf=PATIENT_CPF)
print("Wrap and send - Request Record")
response = client._wrap_and_send(rawPayload)
print("Response: {}".format(response))
```

```
PS C:\sawtooth-example\sawtooth> docker exec -t sawtooth-dev-default bash -c "cd sawtooth/client/ ; python3 dapp_test.py"
Wrap and send - Request Record
Response: {
  "link": "http://rest-api:8008/batch_statuses?id=a0d582614a96ac94cf787789380c4e1060e4b02eff05b0578090b5ece69888b82a7238601f682ded8d0681b69ca303344cf17fee1dbe34bd155113a67a140aa1"
}
PS C:\sawtooth-example\sawtooth>
```

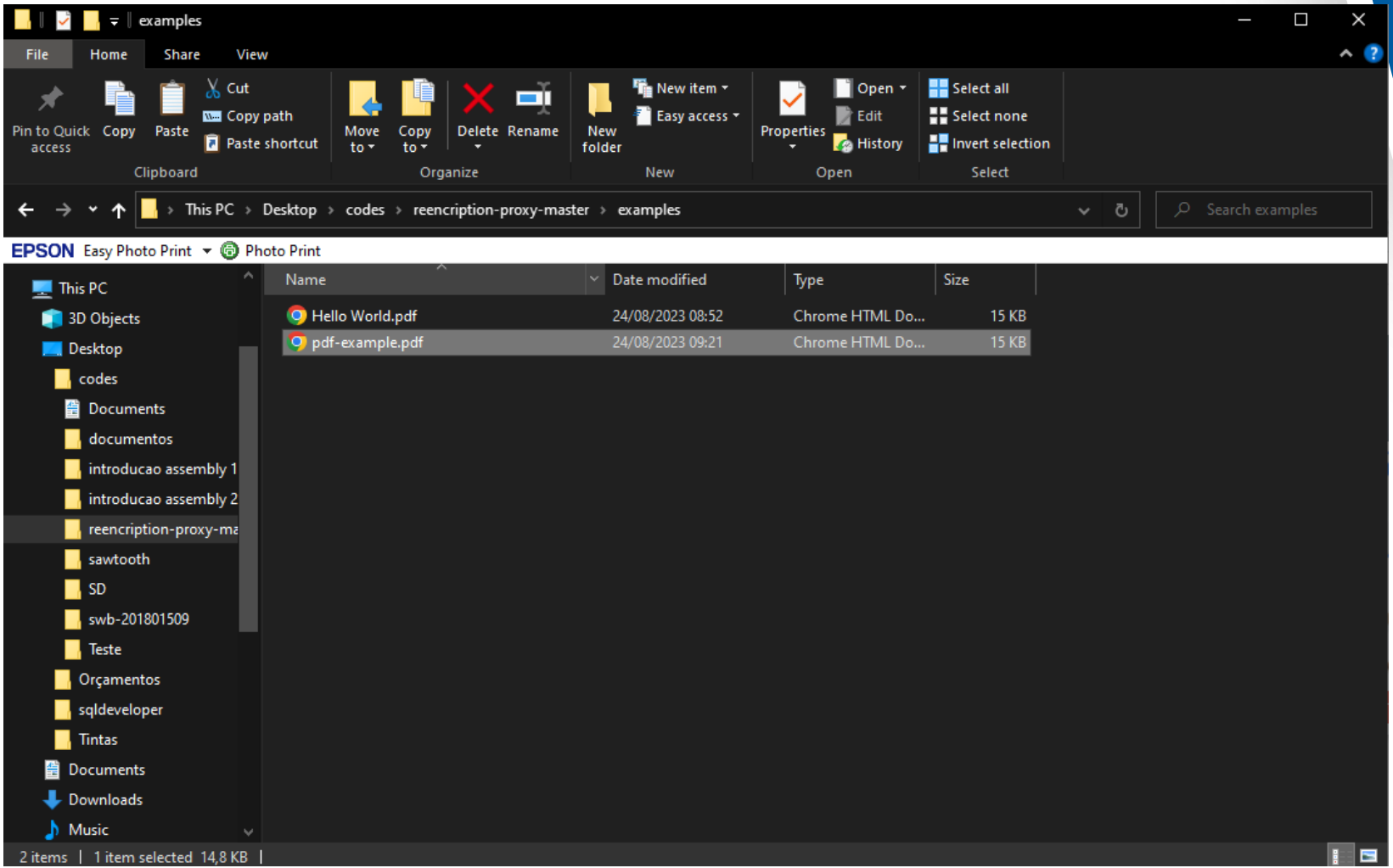
```
Record requested
Name: Joana, CPF: 222.222.222-22, Type: Patient, Records: ID: 1, bundle_hash: BUNDLE_HASH12312312, title: Record Title 1, Requests: --- ID: ID_rEQUEST1
, Doctor_CPF: 111.111.111-11, request_status: 0 ---


Record requested
Name: Joana, CPF: 222.222.222-22, Type: Patient, Records: ID: 1, bundle_hash: BUNDLE_HASH12312312, title: Record Title 1, Requests: --- ID: ID_rEQUEST1
, Doctor_CPF: 111.111.111-11, request_status: 0 ---
```

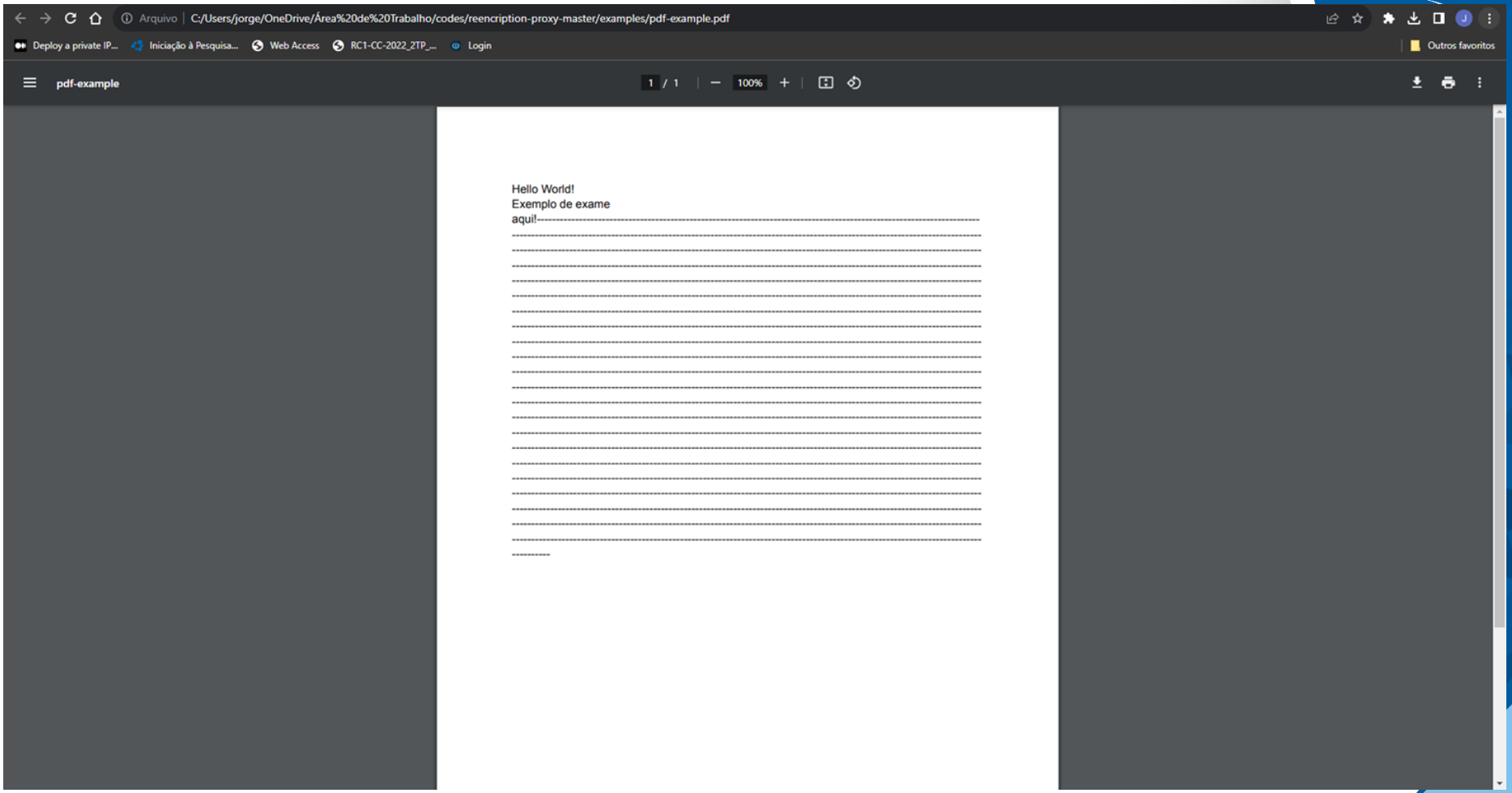

/ | Files | IPFS
IPFS
STATUS
FILES
EXPLORE
PEERS
SETTINGS
UI v2.15.0
Revision fadd900
See the code
Report a bug
QmHash/bafyHash
Browse
Files
4B
FILES
7MiB
ALL BLOCKS
+ Import
Name ↑
Pin Status
Size
No files in this directory. Click the "Import" button to add some.
No files here yet! Add files to your local IPFS node by clicking the Import button above.

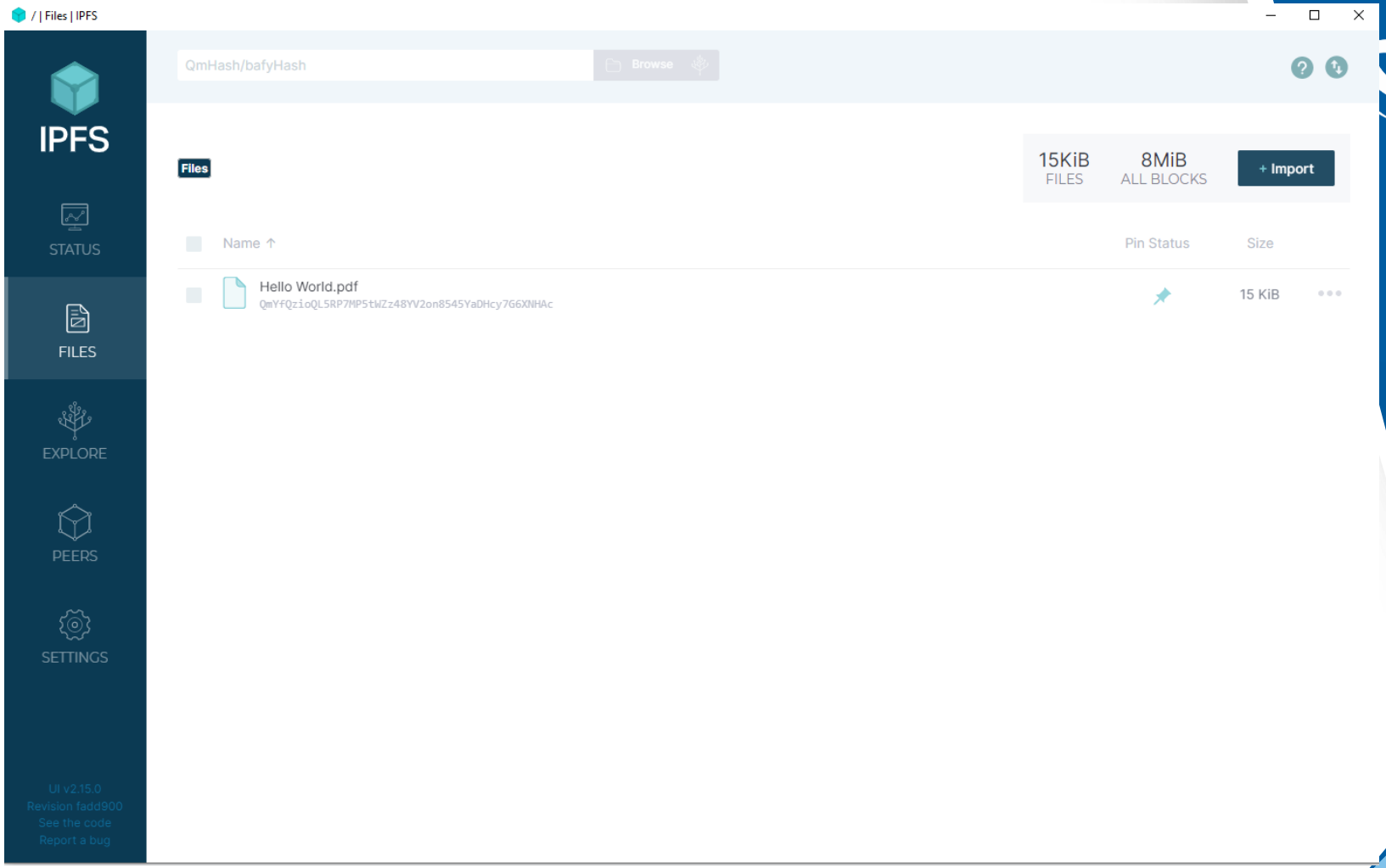

examples
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC › Desktop › codes › reencription-proxy-master › examples
Search examples
EPSON Easy Photo Print
Photo Print
OneDrive - Personal
This PC
3D Objects
Desktop
codes
Documents
documentos
introducao assembly 1
introducao assembly 2
reencription-proxy-ma
sawtooth
SD
swb-201801509
Teste
Orçamentos
sqldeveloper
Tintas
Documents
Downloads
Name
Date modified
Type
Size
Hello World.pdf
24/08/2023 08:52
Chrome HTML Do...
15 KB
1 item

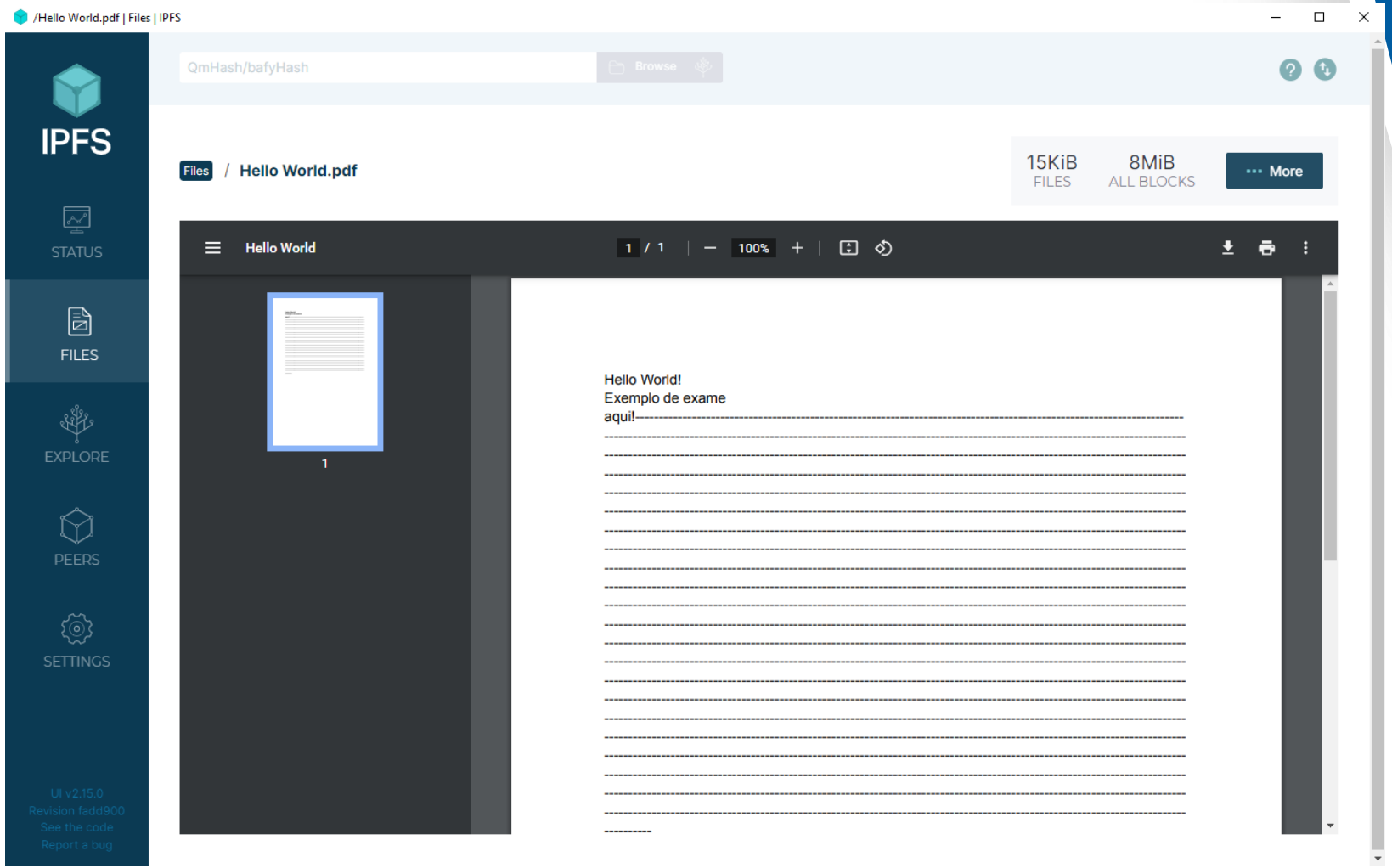
```python
def main():
    with open("./examples/Hello World.pdf", "rb") as file:
        content = file.read()
        create_instance(content)

    content = test_instance()
    with open("./examples/pdf-example.pdf", "wb") as file:
        file.write(content)
    print("Conteudo escrito em um arquivo")

if(__name__=="__main__"):
    main()
```

```
PS C:\Users\jorge\OneDrive\Área de Trabalho\codes\reencription-proxy-master> python setup.py
O Record do paciente foi armazenado na IPFS em QmYfQzioQL5RP7MP5tWZz48YV2on8545YaDHcy7G6XNHAc

Arquivo do Contrato Inteligente foi criado

Arquivo com as chaves do Doutor foi criado

Token obtido do armazenamento do SC

Token decodificado

Cifra e capsula obtidas do IPFS

Conteudo decodificado por reencriptacao

Conteudo escrito em um arquivo
PS C:\Users\jorge\OneDrive\Área de Trabalho\codes\reencription-proxy-master>
```


examples
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC
Desktop
codes
reencription-proxy-master
examples
Search examples
EPSON Easy Photo Print
Photo Print
This PC
3D Objects
Desktop
codes
Documents
documentos
introducao assembly 1
introducao assembly 2
reencription-proxy-ma
sawtooth
SD
swb-201801509
Teste
Orçamentos
sqldeveloper
Tintas
Documents
Downloads
Music
Name
Date modified
Type
Size
Hello World.pdf
24/08/2023 08:52
Chrome HTML Do...
15 KB
pdf-example.pdf
24/08/2023 09:21
Chrome HTML Do...
15 KB
2 items | 1 item selected 14,8 KB


Arquivo | C:/Users/jorge/OneDrive/Área%20de%20Trabalho/codes/reencription-proxy-master/examples/pdf-example.pdf
Deploy a private IP...
Iniciação à Pesquisa...
Web Access
RC1-CC-2022_2TP_...
Login
Outros favoritos
pdf-example
1 / 1
100%
Hello World!
Exemplo de exame
aqui!----------


/ | Files | IPFS
IPFS
QmHash/bafyHash
Browse
STATUS
FILES
EXPLORE
PEERS
SETTINGS
Files
15KiB
FILES
8MiB
ALL BLOCKS
+ Import
Name
Pin Status
Size
Hello World.pdf
QmYfQzioQL5RP7MP5tWZz48YV2on8545YaDHcy7G6XNHAc
15 KiB
UI v2.15.0
Revision fadd900
See the code
Report a bug


/Hello World.pdf | Files | IPFS
IPFS
QmHash/bafyHash
Browse
STATUS
FILES
EXPLORE
PEERS
SETTINGS
Files / Hello World.pdf
15KiB FILES
8MiB ALL BLOCKS
More
Hello World
1 / 1
100%
1
Hello World!
Exemplo de exame
aqui!----------
UI v2.15.0
Revision fadd900
See the code
Report a bug

Referências

Referências Bibliográficas

Aplicação para armazenamento seguro e descentralizado de registros médicos usando Blockchain e IPFS

[1] Madine, M. M., Al-Ayyoub, M., Jararweh, Y., & Al-Zoubi, H. (2020). Blockchain for Giving Patients Control Over Their Medical Records. IEEE Access, 8, 217487-217500.

[2] Kumar, S., Bharti, A. K., & Amin, R. (2021). Decentralized secure storage of medical records using Blockchain and IPFS: A comparative analysis with future directions. Security and Privacy, 4(5), e173.

[3] Kumar, S., Bharti, A. K., & Amin, R. (2021). Secure and Scalable Decentralized Supply Chain Management Using Blockchain and IPFS. IEEE Communications Magazine, 59(10), 68-74.

Obrigado!

UFG
UNIVERSIDADE FEDERAL DE GOIÁS