



Certificazione trustless di documenti tramite NFT minting su Blockchain



Candidato: Tommaso Mangiavacchi

Relatrice: Laura Emilia Maria Ricci

Tutor Aziendale: Luca Secci



Advinser

Advinser Srl è un'azienda di realizzazione software con sede ad Asciano (SI), fondata nel 2020.

I principali servizi offerti sono:

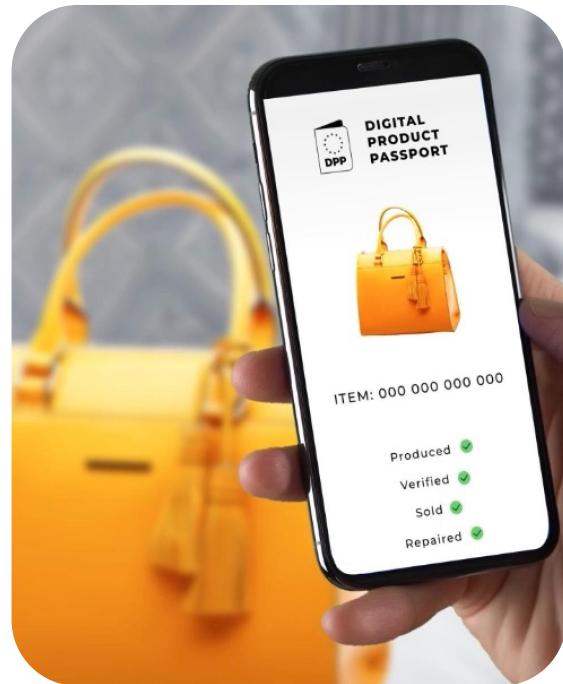
- Piattaforme SAAS
- System Integration
- Software Tailor Made



Obiettivo del progetto

Lo scopo del progetto è fornire alle aziende del fashion di lusso un **Digital Product Passport** (DPP) per i loro prodotti che assicuri:

- Autenticità e Proprietà dei prodotti
- Tracciabilità completa della supply chain
- Trasparenza dei dati tramite Blockchain

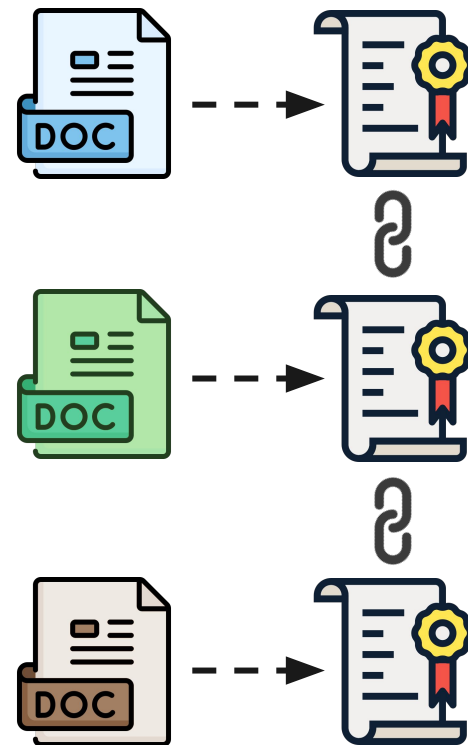


Realizzazione del DPP

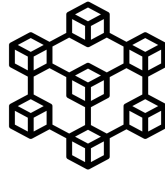
Un DPP è un insieme di documenti in cui vengono descritte le **fasi di produzione** e altre informazioni.

La soluzione adottata consiste in:

- **Caricare** i documenti su una rete decentralizzata
- **Certificare** i documenti dei prodotti su blockchain
- **Associare** ad ogni certificato un token
- **Concatenare** i certificati riguardanti un prodotto



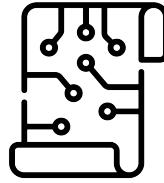
Blockchain



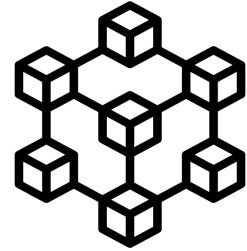
È una rete decentralizzata di nodi che collaborano per emettere e verificare transazioni. Alcune proprietà fondamentali sono:

- **Decentralizzazione** : La gestione è distribuita tra molti nodi indipendenti senza una autorità centralizzata.
- **Immutabilità** : un blocco di transazioni, una volta aggiunto alla blockchain, diventa permanente e inalterabile.
- **Trasparenza** : Tutte le transazioni sono visibili a tutti i nodi della rete, garantendo fiducia e verifica indipendente.

Smart Contract



- **Programmi** situati su una blockchain.
- **Immutabili** una volta distribuiti.
- **Verificabili** da tutti i partecipanti alla rete.
- **Eseguiti** su tutti i nodi della blockchain.
- **Gas** è usato per misurare e pagare le risorse computazionali.



NFT

- **Risorse digitali uniche** memorizzate su una blockchain.
 - Certificano **l'autenticità** e la **proprietà** di un oggetto digitale.
- Basati sullo standard **ERC-721** degli smart contract
 - Implementazione fornita dalla libreria **OpenZeppelin**.

Questo li rende ideali per rappresentare:

- Opere d'arte
- Musica
- Documenti
- Altri contenuti digitali

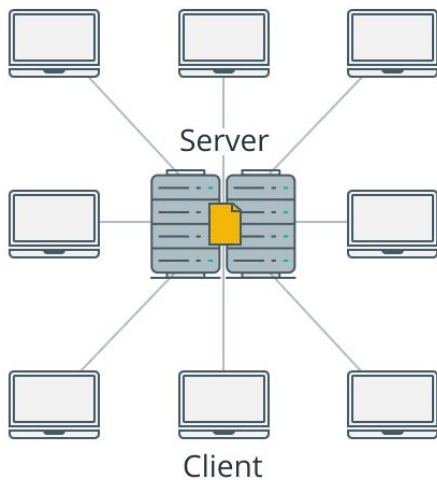


IPFS

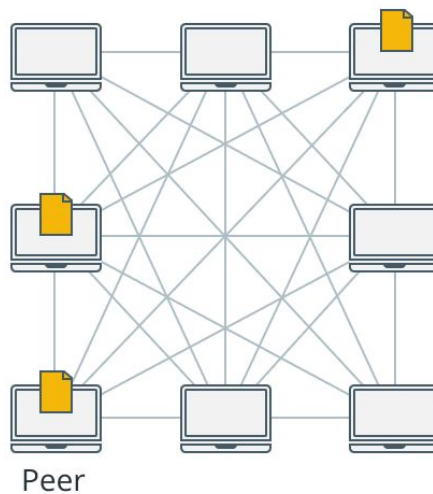


Rete peer-to-peer decentralizzata per l'archiviazione dei dati.

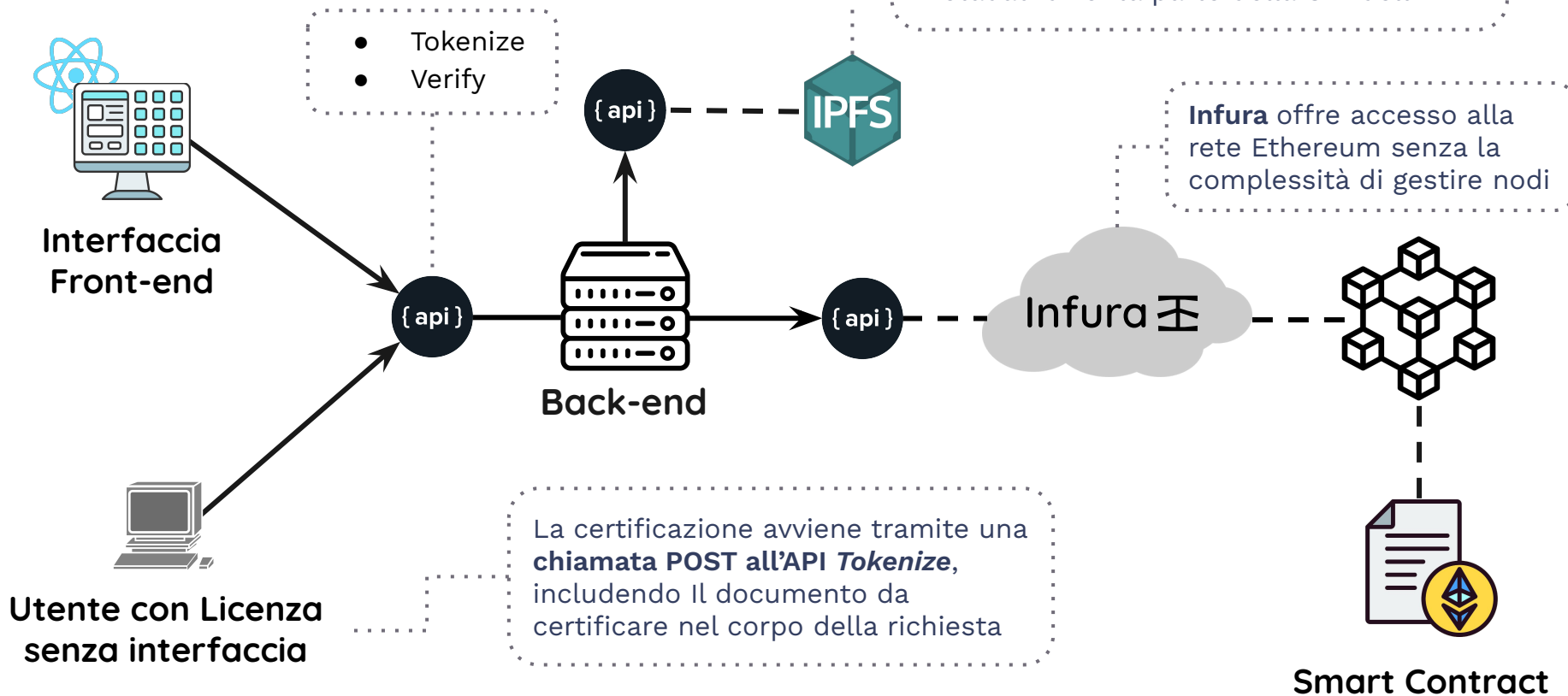
Client-server model HTTP



Peer-to-peer model IPFS



The Big Picture



Implementazione Smart Contract



- Sviluppato in **Solidity**.
- Sviluppato e collaudato con **Hardhat**.
- Implementazione di **ERC-721A**.
- Testing sulla testnet di **Linea**.



Solidity

Linguaggio di programmazione per smart contract



Hardhat

Strumento di sviluppo per smart contract su Ethereum



Linea

Blockchain layer 2 utilizzata per migliorare scalabilità e ridurre costi

Operazioni dello smart contract



Certificazione

Generazione di un NFT che contiene, nei metadati, un riferimento al documento caricato su IPFS



Aggiornamento

Certificazione, Burn dell'NFT precedente e concatenazione delle certificazioni tramite inserimento nello storico



Verifica

Restituisce una ricevuta relativa all'**Hash** del documento passato come parametro

Frammento dello Smart Contract

```
1 function safeMint(
2     address toAddress,
3     bytes32 tokenUri,
4     bytes32 cid,
5     uint256 obsoleteTknId
6 ) external onlyOwner {
7     unchecked {
8         uint256 index;
9         uint256 __nextTokenId = __nextTokenId;
10        uint256 cidItem = cidMap[cid];
11        if (cidItem != 0) {
12            revert documentAlreadyCertified(cidItem);
13        }
14        if (obsoleteTknId > 0) {
15            _burn(obsoleteTknId);
16            index = tokenMap[obsoleteTknId].index;
17            if (index == 0) {
18                tokenMap[obsoleteTknId].index = obsoleteTknId;
19                tokenMap[__nextTokenId].index = obsoleteTknId;
20                historyMap[obsoleteTknId].push(__nextTokenId);
21            } else {
22                tokenMap[__nextTokenId].index = index;
23                historyMap[index].push(__nextTokenId);
24            }
25        }
26        _safeMint(toAddress, __nextTokenId);
27        cidMap[cid] = __nextTokenId;
28        tokenMap[__nextTokenId].cid = cid;
29        tokenMap[__nextTokenId].uri = tokenUri;
30        tokenMap[__nextTokenId].timestamp = block.timestamp;
31        ++__nextTokenId;
32    }
33 }
```

```
1 function verify(
2     bytes32 cid
3 ) public view returns (CompoundReceipt memory receipt) {
4     uint256 tokenId = cidMap[cid];
5     uint256 tokenId_history;
6     uint256 index;
7     uint256 length;
8     uint256 j;
9     DecodedHistory[] memory _history;
10    index = tokenMap[tokenId].index;
11    if (index != 0) {
12        length = historyMap[index].length;
13        _history = new DecodedHistory[](length + 1);
14        _history[0] = DecodedHistory({
15            timestamp: tokenMap[index].timestamp,
16            tokenId: index,
17            cid: bytes32HexString(tokenMap[index].cid),
18            tokenUri: tokenURI(index)
19        });
20        for (j = 0; j < length; ++j) {
21            tokenId_history = historyMap[index][j];
22            _history[j + 1] = DecodedHistory({
23                timestamp: tokenMap[tokenId_history].timestamp,
24                tokenId: tokenId_history,
25                cid: bytes32HexString(tokenMap[tokenId_history].cid),
26                tokenUri: tokenURI(tokenId_history)
27            });
28        }
29    }
30    receipt = CompoundReceipt({
31        timestamp: tokenMap[tokenId].timestamp,
32        cid: bytes32HexString(cid),
33        tokenUri: tokenURI(tokenId),
34        tokenId: tokenId,
35        history: _history
36    });
37    return receipt;
38 }
```

Ottimizzazioni Gas



- Uso di Byte32 per contenere gli hash 32-byte sha-256.
- Minimizzazione letture/scritture dallo stato della blockchain (*Storage*).
 - Caching variabili.
- Creazione di cidMap per mappare gli hash URI con i tokenID.
- Controllo di integrità e annullamento delle operazioni per evitare sprechi.
- Uso di ERC-721A per mintare più NFT in una singola operazione.
- Memorizzazione dei tokenID bruciati in array uint64, salvando 4 tokenID in uno uint256 per ridurre le scritture.


→ Ridurre il gas utilizzato è **cruciale** per abbattere i costi di esecuzione del contratto.

Analisi dei Costi

| | ERC721 (dedicato) | ERC721A (condiviso) |
|----------------|----------------------|------------------------|
| Certificazione | 0.03 € | 0.025 € |
| Aggiornamento | 0.05 € | 0.035 € |

Versione Finale

- **ERC721A (condiviso):**
 - Troppo complesso
 - Scarsa manutenibilità
 - Mescolanza di dati tra aziende.

- 
- **ERC721 (dedicato):**
 - Più Semplice
 - Buona manutenibilità
 - Isolamento dei dati

Grazie per l'attenzione !

Laurea in Informatica
Università di Pisa

Appello di laurea del 12/07/2024

