



KRYPTOAUTH

Corso di Laurea Magistrale in
Sicurezza Informatica

Prof.:
Christiancarmine Esposito
Alfredo De Santis

Presentazione di:
Alberto Montefusco
Mat. 0522501498

**ANNO ACCADEMICO
2022/2023**

CONTENUTI

01

INTRODUZIONE

La Blockchain e il
mondo NFT

KRYPTOAUTH

Cos'è KryptoAuth, analisi
dei requisiti principali

02

03

SMART CONTRACT

Tecnologie utilizzate,
gestione dello smart
contract

SOLUZIONI SICURE

Sicurezza adottata
nella Web DApp

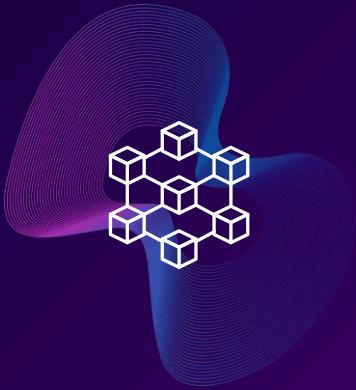
04

01

INTRODUZIONE

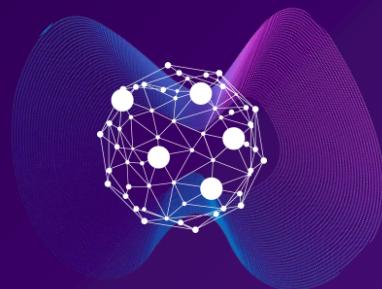
LA RIVOLUZIONE DEL WEB3

IL WEB 3



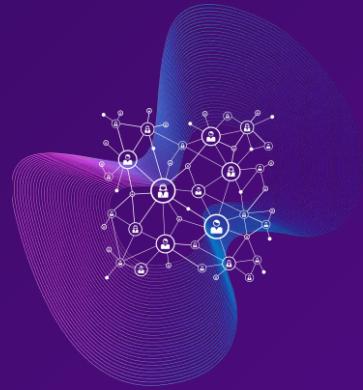
BLOCKCHAIN

Sistema decentralizzato
che permette la
condivisione trasparente di
informazioni all'interno di
una rete aziendale



SERVER

Le applicazioni odierne sono
in esecuzione su server
centralizzati che hanno il
compito di gestire e
conservare dati sensibili



NODI DELLA RETE

Vari elaboratori in
grado di comunicare
con gli altri dispositivi
che fanno parte della
rete

CARATTERISTICHE PRINCIPALI

DECENTRALIZZAZIONE

Trasferimento del controllo e del processo decisionale da un'entità centralizzata ad una rete distribuita

CONSENSO

È possibile registrare nuove transazioni soltanto quando la maggioranza dei partecipanti alla rete dà il proprio consenso



IMMUTABILITÀ

Qualcosa che non può essere modificato o alterato. Se un registro di transazione include un errore, è necessario aggiungere una nuova transazione per annullarlo, ed entrambe le transazioni sono visibili alla rete

NFTs

Certificati “di proprietà” su opere digitali utilizzati dagli utenti per sostituire dati, beni, diritti con un unico asset digitale per usufruire di servizi resi sicuri dalla Blockchain

Gli acquisti in NFT avvengono mediante la Blockchain **Ethereum**, in cui vengono deployati rigorosi smart contracts.



02

KRYPTOAUTH

REQUISITI PRINCIPALI

E-COMMERCE CENTRALIZZATI

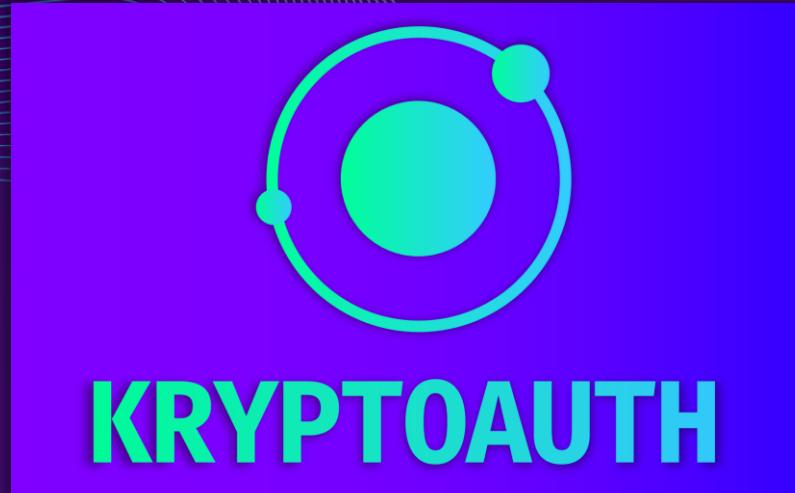


Gli utenti possono acquistare diversi coupon per ottenere vari sconti in diverse categorie di acquisto per essere incentivati a spendere il più possibile per uno specifico brand.

Poco sicuro in quanto nulla certifica la proprietà o l'autenticità di uno specifico coupon acquistato o usato da un utente.

SISTEMA PROPOSTO

Gestione di un Marketplace di NFTs e definizione di ruoli specifici per implementare una sicura politica degli accetti tramite la Blockchain Ethereum



DEFINIZIONE DEI RUOLI



- Gestione degli accessi
- Disattivare il proprio account
- Gestione di un Marketplace di NFTs

Ogni amministratore ha il pieno controllo dei propri dati personali senza dover dipendere da terzi

DEFINIZIONE DEI RUOLI

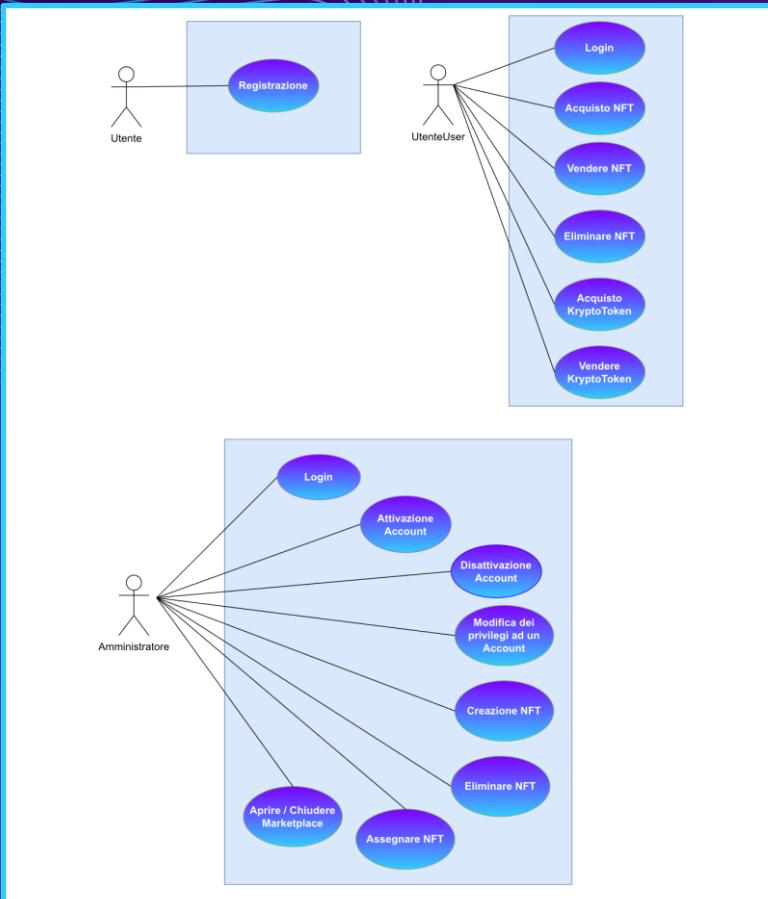


- Registrazione / Login in KryptoAuth
- Acquistare, usare e vendere NFTs posseduti
- Acquistare e riconvertire in ethere KryptoToken

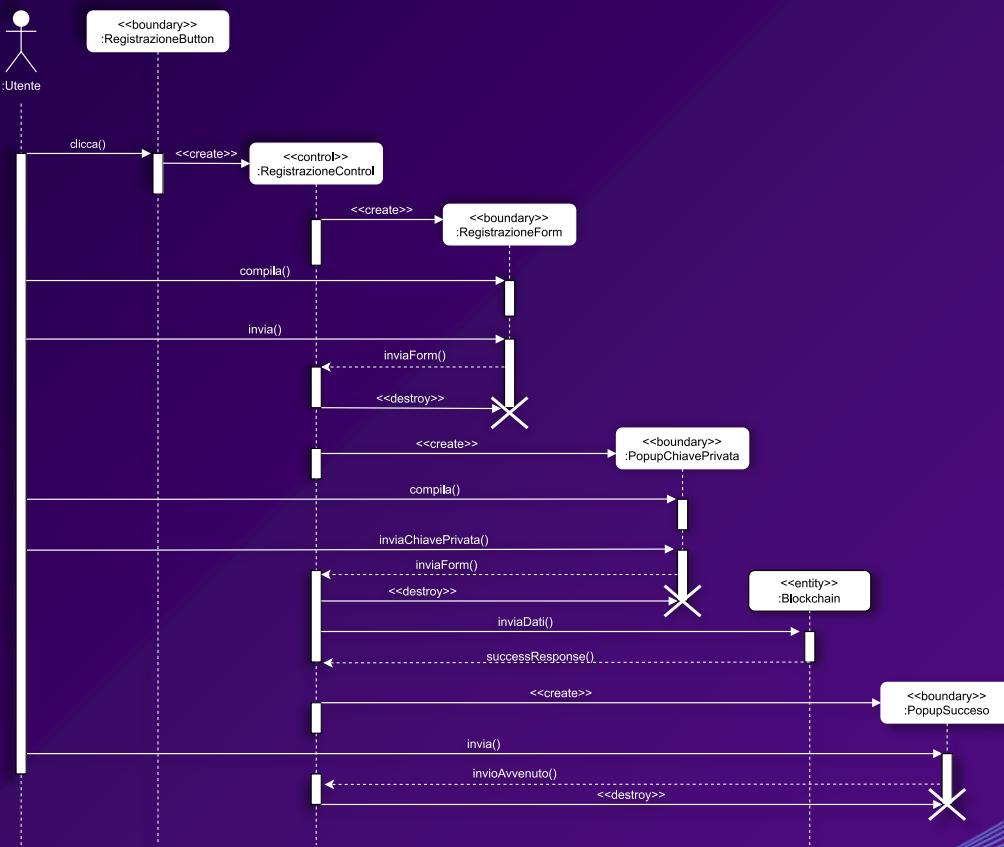
Ogni utente ha il pieno controllo dei propri dati personali senza dover dipendere da terzi

USE CASE MODEL

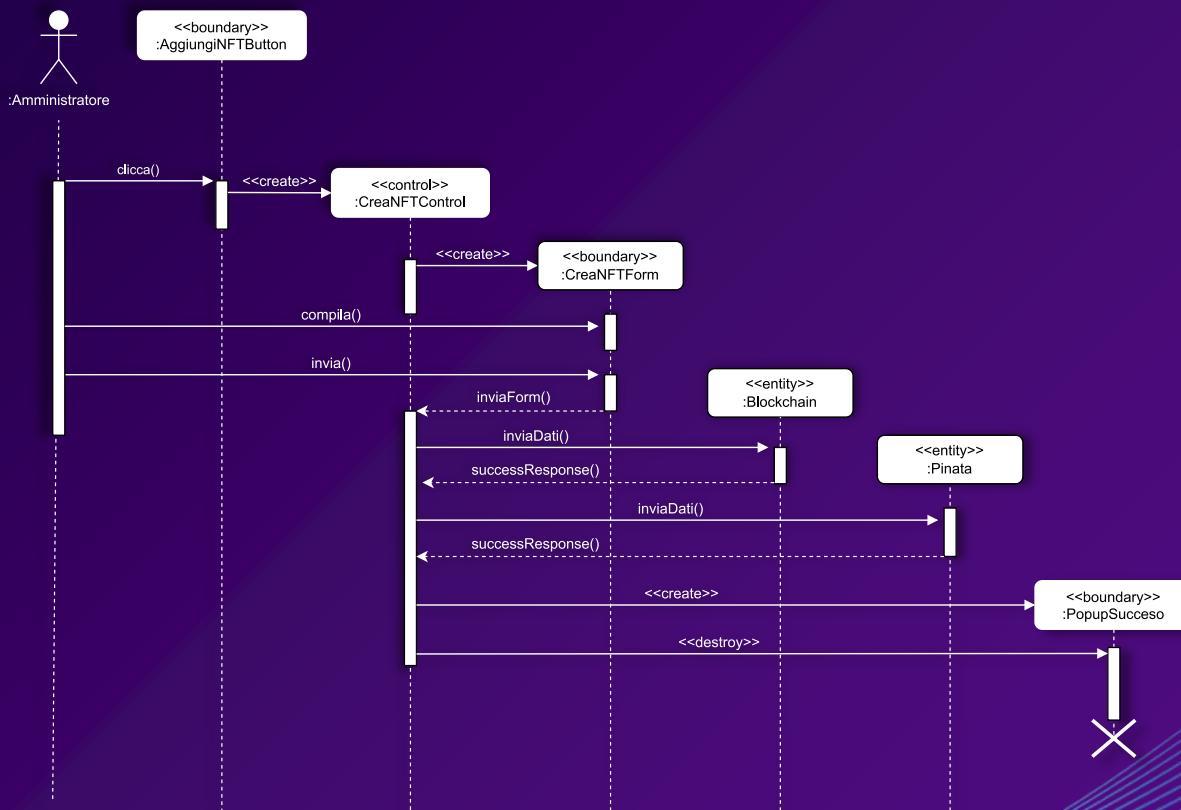
Mostriamo lo Use Case model
di tutte le funzionalità che
KryptoAuth garantisce agli
utenti



REGISTRAZIONE NUOVO UTENTE



CREAZIONE DI UN NFT



03

SMART CONTRACT

GESTIONE DELLO SMART CONTRACT

TECNOLOGIE UTILIZZATE

GANACHE

Blockchain di test
basata su Ethereum

METAMASK

Wallet crittografico e
gateway per Web DApp

WEB3J e WEB3js

Interfacciamento tra
MetaMask e la
Blockchain Ganache



TRUFFLE

Framework per il deploy
degli Smart Contracts

SOLIDITY

Linguaggio degli
Smart Contracts

SPRING BOOT

Framework per
il backend

PINATA

Pinata un gateway dedicato
che permette di reperire i
contenuti più velocemente dai
nodi IPFS risparmiando
larghezza di banda e tempo



IMPLEMENTAZIONE AUTHENTICATION.SOL

Authentication.sol è uno smart contract scritto in Solidity reso sicuro e robusto tramite la libreria OpenZeppelin per la definizione dei ruoli assunta dagli utenti: "Admin" o "User"

```
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/access/AccessControl.sol";

contract Authentication is AccessControl {
    bytes32 public constant USER_ROLE = keccak256("USER");

    struct User {
        address addr;
        string name;
        bytes32 password;
    }

    mapping(address => User) user;

    constructor() {
        _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
        _setRoleAdmin(USER_ROLE, DEFAULT_ADMIN_ROLE);
    }

    /* Ristretto ai membri che hanno l'admin role. */
    modifier onlyAdmin(){
        require(isAdmin(msg.sender), "Restricted to admins.");
       _;
    }

    /* Ristretto ai membri che hanno l'user role. */
    modifier onlyUser(){
        require(isUser(msg.sender), "Restricted to users.");
       _;
    }
}
```

IMPLEMENTAZIONE AUTHENTICATION.SOL

Solo l'amministratore può assegnare o revocare un ruolo ad un utente, mentre, l'admin può rinunciare ai suoi privilegi solo se egli stesso ne rinuncia

```
/* Aggiunge un account con il ruolo di user (lo possono fare solo gli admin). */
function addUser(address account) public virtual onlyAdmin returns (bool){
    if(!hasRole(USER_ROLE, account)){
        grantRole(USER_ROLE, account);
        return true;
    } return false;
}

/* Aumenta i privilegi ad un account assegnandogli il ruolo di admin e revocando
 * quello di user (lo possono fare solo gli admin). */
function addAdmin(address account) public virtual onlyAdmin returns (bool){
    if(!hasRole(DEFAULT_ADMIN_ROLE, account)){
        removeUser(account);
        grantRole(DEFAULT_ADMIN_ROLE, account);
        return true;
    } return false;
}

/* Rimuove un account dal ruolo di user (lo possono fare solo gli admin). */
function removeUser(address account) public virtual onlyAdmin returns (bool){
    if(hasRole(USER_ROLE, account)){
        revokeRole(USER_ROLE, account);
        return true;
    } return false;
}

/* Un admin rinuncia ad esserlo. */
function renounceAdmin(address account) public virtual onlyAdmin returns (bool){
    if(hasRole(DEFAULT_ADMIN_ROLE, account)){
        renounceRole(DEFAULT_ADMIN_ROLE, account);
        return true;
    } return false;
}
```

IMPLEMENTAZIONE KRYPTONFT.SOL

Kryptonft.sol è uno smart contract scritto in Solidity per la gestione di un Marketplace di NFTs in ERC-1155

```
pragma solidity ^0.8.17;

import "@openzeppelin/contracts/token/ERC1155/ERC1155.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "OpenZeppelin/contracts/utils/Strings.sol";
import "@openzeppelin/contracts/security/ReentrancyGuard.sol";
import "BokkyPooBahsDateTimeLibrary/contracts/BokkyPooBahsDateTimeLibrary.sol";
import "./Authentication.sol";

contract Kryptonft is ERC1155, Authentication, ReentrancyGuard {
    using Counters for Counters.Counter;
    Counters.Counter private _nftCount;

    uint256 private priceToken = 0.04 ether; // KryproToken (KT) price
    bool private isSaleActive = false; // opened or closed marketplace

    mapping(uint256 => NFT) private _idToNFT; // associates the unique tokenId with an NFT structure

    struct NFT {
        uint256 tokenId;
        address seller;
        address owner;
        string name;
        string url;
        string category;
        string description;
        uint256 price;
        uint256 validUntil;
        uint256 sale;
        bool sold;
        bool burned;
    }

    constructor() ERC1155("") {
    }

    function supportsInterface(bytes4 interfaceId) public view virtual override(ERC1155, AccessControl) returns (bool) {
        return super.supportsInterface(interfaceId);
    }
}
```

IMPLEMENTAZIONE KRYPTONFT.SOL

Operazioni riservate
all'amministratore di
KryptoAuth:

- Creazione di un NFT;
- Eliminazione di un NFT;
- Assegnazione di un NFT

```
/* Only Admin can create a new NFT in Blockchain */
function mintNft(
    string memory _name,
    string memory _category,
    string memory _description,
    string memory _url,
    uint256 _price,
    uint256 _dayslimit,
    uint256 _sale
) public onlyAdmin {
    require(_dayslimit > 0 && _price > 0, "Days limit and price must be greater than zero");
    require(_sale > 4 && _sale < 51, "Sale error");

    _nftCount.increment();
    uint256 newTokenId = _nftCount.current();
    require(balanceOf(msg.sender, newTokenId) == 0, "NFT already exist");

    _mint(msg.sender, newTokenId, 1, bytes(_name));
    _idToNFT[newTokenId] = NFT(
        newTokenId,
        msg.sender,
        msg.sender,
        _name,
        _url,
        _category,
        _description,
        _price,
        block.timestamp + (_daysLimit * 1 days),
        _sale,
        false,
        false
    );
}

/* Only Admin can burn the NFT that he didn't sell */
function burnNft(uint256 _tokenId) public onlyAdmin {
    require(_tokenId > 0, "This is not an NFT");
    require(balanceOf(msg.sender, _tokenId) > 0, "This address not have this NFT");

    NFT storage nft = _idToNFT[_tokenId];
    require(!nft.sold, "Token already sold");
    require(nft.owner == msg.sender && nft.owner == nft.seller, "NFT already bought");

    nft.burned = true;
    _burn(msg.sender, _tokenId, 1);
}

/* Only Admin can transfer the NFT sold to account which bought that */
function assignNft(uint256 _tokenId, address _addr) public onlyAdmin {
    require(_tokenId > 0, "This is not an NFT");

    NFT storage nft = _idToNFT[_tokenId];
    require(balanceOf(msg.sender, _tokenId) > 0, "This address not have this NFT");
    require(isUser(_addr), "This address isn't an User");
    require(nft.owner == _addr, "User address wrong");
    require(!nft.sold, "NFT already sold");

    nft.sold = true;
    safeTransferFrom(msg.sender, _addr, _tokenId, 1, bytes(nft.name));
}
```

IMPLEMENTAZIONE KRYPTONFT.SOL

Operazioni riservate ad un Utente di KryptoAuth:

- Comprare, vendere, usare, eliminare un NFT;
- Comprare o vendere un KT

```
/* Assigns the ether to the User that sold his coins */
function sellNFT(uint256 _amounts) public payable onlyUser nonReentrant {
    require(_amounts > 0, "Amounts have not be 0");
    require(balanceOf(msg.sender, 0) >= _amounts, "Not enough token");

    _burn(msg.sender, 0, _amounts);
    payable(msg.sender).transfer(_amounts * priceToken);
}

/* Assigns the coins to the User that sold his NFT and transfers the NFT to seller */
function sellNFTUser(uint256 _tokenId) public onlyUser {
    require(_tokenId > 0, "This is not an NFT");

    NFT storage nft = _idToNFT[_tokenId];
    require(balanceOf(msg.sender, _tokenId) > 0, "This address not have this NFT");
    require(isValid(_tokenId), "NFT is expired");

    if(isAdmin(nft.seller)){
        nft.sold = false;
        nft.owner = nft.seller;
        safeTransferFrom(msg.sender, nft.seller, _tokenId, 1, bytes(nft.name));
    }
    else{
        nft.burned = true;
        _burn(msg.sender, _tokenId, 1);
    }
    _mint(msg.sender, 0, nft.price, "KryptoToken");
}
```

```
/* User burn his NFT and transfers its to seller if he still has the Admin role */
function useNFT(uint256 _tokenId) public onlyUser {
    require(_tokenId > 0, "This is not an NFT");

    NFT storage nft = _idToNFT[_tokenId];
    require(balanceOf(msg.sender, _tokenId) > 0, "This address not have this NFT");
    require(isValid(_tokenId), "NFT is expired");

    if(isAdmin(nft.seller)){
        nft.sold = false;
        nft.owner = nft.seller;
        safeTransferFrom(msg.sender, nft.seller, _tokenId, 1, bytes(nft.name));
    }
    else{
        nft.burned = true;
        _burn(msg.sender, _tokenId, 1);
    }

    /* User burn his NFT expired */
    function burnNFTUser(uint256 _tokenId) public onlyUser {
        require(_tokenId > 0, "This is not an NFT");

        NFT storage nft = _idToNFT[_tokenId];
        require(balanceOf(msg.sender, _tokenId) > 0, "This address not have this NFT");

        nft.burned = true;
        _burn(msg.sender, _tokenId, 1);
    }
}
```

```
***** User Functions *****/
/* Only User can buy KryptoToken (KT) */
function buyKT(uint256 _amounts) public payable onlyUser nonReentrant {
    require(_amounts > 0, "Amounts must be greater than zero");
    require(msg.value == priceToken * _amounts, "WRONG: Not enough money sent");

    _mint(msg.sender, 0, _amounts, "KryptoToken");
}

/* User buy NFTs and the seller gets paid; the User has to wait admin to assign him the NFT */
function buyNFT(uint256 _tokenId) public payable onlyUser nonReentrant {
    require(isSaleActive, "The marketplace is closed");
    require(_tokenId > 0, "This is not an NFT");

    NFT storage nft = _idToNFT[_tokenId];
    require(!nft.burned, "NFT doesn't exist");
    require(balanceOf(msg.sender, 0) >= nft.price, "Not enough tokens");
    require(nft.owner == nft.seller, "NFT already sold");
    require(isAdmin(nft.seller), "Seller is not an Admin");
    require(isValid(_tokenId), "NFT is expired");

    nft.owner = msg.sender;
    _burn(msg.sender, 0, nft.price);
    payable(nft.seller).transfer(priceToken * nft.price);
}
```

DEPLOY SMART CONTRACT

```
Compiling your contracts...
=====
> Everything is up to date, there is nothing to compile.

Starting migrations...
=====
> Network name:    'development'
> Network id:     1563307888769
> Block gas limit: 0x6691b7

1_initial_migration.js
=====

Deploying 'Migrations'
=====
> transaction hash: 0xbff9970dd3509214170768424a1c74f94fe2c91cafce47f81de71776a524e5ddc
> Blocks: 0
> contract address: 0xa089179CEA8153e7e28e8AF120ba8A2b30E58354
> block number: 1
> block timestamp: 1563307902
> account: 0x1862C73CadFa924B8787A0c66a46daD346257158
> balance: 99.99430184
> gas used: 284908
> gas price: 20 gwei
> value sent: 0 ETH
> total cost: 0.00569816 ETH
```

Deploy dello Smart Contract con truffle digitando nel terminale:

- **truffle compile**, per verificare la presenza di errori sintattici;
- **truffle migrate**, per deployare lo Smart Contract sulla Blockchain Ganache

TRADUZIONE SMART CONTRACT

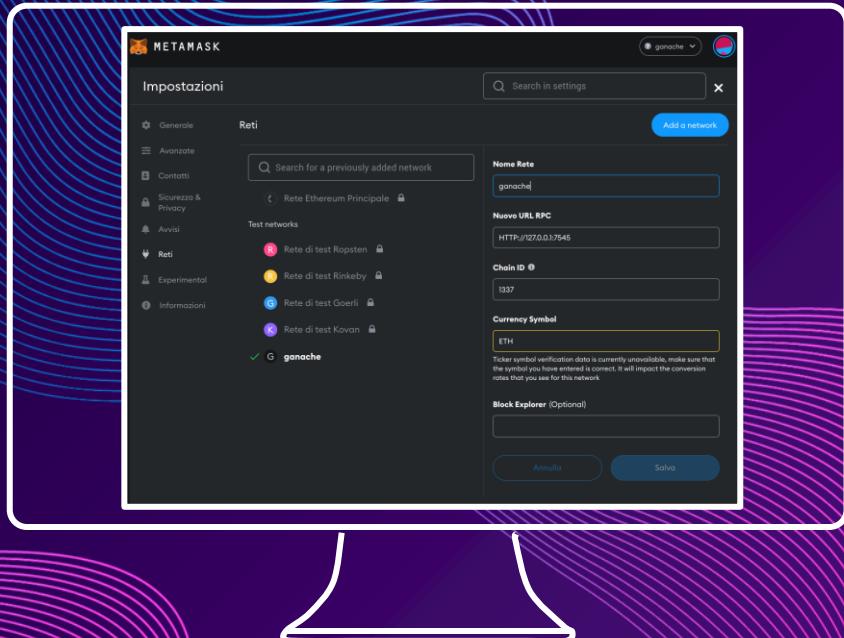
```
alberto@alberto-ASUS:~$ solcjs /home/alberto/Documents/Github/Blockchain-Authentication/KryptoAuth  
/src/main/'smart contract'/contracts/Authentication.sol --bin --include-path node_modules/ --base-  
path . --abi --optimize -o /home/alberto/Documents/Github/Blockchain-Authentication/KryptoAuth/src  
/main/resources/solidity
```

```
alberto@alberto-ASUS:~$ web3j generate solidity -b ./src/main/resources/solidity/Authentication.bi  
n -a ./src/main/resources/solidity/Authentication.abi -o ./src/main/java -p lt.unisa.KryptoAuth.co  
ntracts
```

Lo Smart Contract è stato tradotto in una classe Java tramite:

- il compilatore **solcjs**, per la generazione dei file KryptoNFT.abi e KryptoNFT.bin;
- **Web3j**, per la creazione della classe KryptoNFT.java

CONFIGURAZIONE DI UNA NUOVA RETE



Nuova rete configurata sul wallet MetaMask:

- **Nome rete:** Ganache
- **URL RPC:** HTTP://127.0.0.1:7545
- **Chain ID:** 1337
- **Currency Symbol:** ETH

04

SOLUZIONI SICURE SICUREZZA ADOTTATA

OVERFLOW / UNDERFLOW

Gli attacchi di Overflow e Underflow sono respinti mediante l'utilizzo della versione 0.8.17 del compilatore di Solidity



TIMESTAMP

Il timestamp utilizzato non è pericoloso dato che le operazioni effettuate su di esso riguardano giorni e non secondi



ATTACCO DEL RIENTRO

Aggiorniamo lo stato prima di effettuare l'invio di denaro quando viene creato o venduto un NFT ed è usato il modificatore nonReentrant() di OpenZeppelin che impedisce a un contratto di chiamare se stesso



**GRAZIE
PER
L'ATTENZIONE**

