



Utilizzo dei wallet e programmazione con Solidity

Y2Y: BLOCKCHAIN E SMART CONTRACT

Organizzatori: Massimiliano Sala, Andrea Gangemi e Christopher Spennato

Speaker: Christopher Spennato

23 e 30 Novembre, 2023

Overview iniziale

MetaMask

User experience

MetaMask SDK

Solidity

Basi di Solidity

Overview iniziale

Un **wallet** su *blockchain* è un portafoglio digitale che un utente può utilizzare per gestire i propri *token* e le proprie criptovalute.

Un **wallet** su *blockchain* è un portafoglio digitale che un utente può utilizzare per gestire i propri *token* e le proprie criptovalute.

Osservazione

In realtà, un *wallet* non è da intendersi nel senso classico di portafoglio, ma è solo una cassaforte sicura per salvare la propria chiave privata.

Un **wallet** su *blockchain* è un portafoglio digitale che un utente può utilizzare per gestire i propri *token* e le proprie criptovalute.

Osservazione

In realtà, un *wallet* non è da intendersi nel senso classico di portafoglio, ma è solo una cassaforte sicura per salvare la propria chiave privata.

Alcuni esempi:

- *MetaMask*;
- *OKX Wallet*;
- *Trust Wallet*.

MetaMask

Di tutti i *wallet* esistenti, ad oggi quello più diffuso e quindi meglio mantenuto è *MetaMask*.

Di tutti i *wallet* esistenti, ad oggi quello più diffuso e quindi meglio mantenuto è *MetaMask*.

- È un'estensione browser (*web wallet*) o un'applicazione per cellulari (*mobile wallet*);

Di tutti i *wallet* esistenti, ad oggi quello più diffuso e quindi meglio mantenuto è *MetaMask*.

- È un'estensione browser (*web wallet*) o un'applicazione per cellulari (*mobile wallet*);
- Si può usare per interagire con la *blockchain* Ethereum in due modi: sia direttamente, sia attraverso Dapp.

- *MetaMask* è stata sviluppata nel 2016 dalla *ConsenSys*.

- *MetaMask* è stata sviluppata nel 2016 dalla *ConsenSys*.
- Nata inizialmente come estensione browser per Chrome e Firefox.

- *MetaMask* è stata sviluppata nel 2016 dalla *ConsenSys*.
- Nata inizialmente come estensione browser per Chrome e Firefox.
- Per fronteggiare versioni malevole, nel 2020 venne rilasciata la versione ufficiale per iOS e Android.

Versione browser:

Eseguire la procedura di installazione di un nuovo *wallet* o di *import* di un *wallet* già esistente come mostrato.

Versione browser:

Eseguire la procedura di installazione di un nuovo *wallet* o di *import* di un *wallet* già esistente come mostrato.

Versione mobile:

Installare l'applicazione e seguire la procedura guidata in maniera analoga a quanto visto per il web.

Nuova installazione

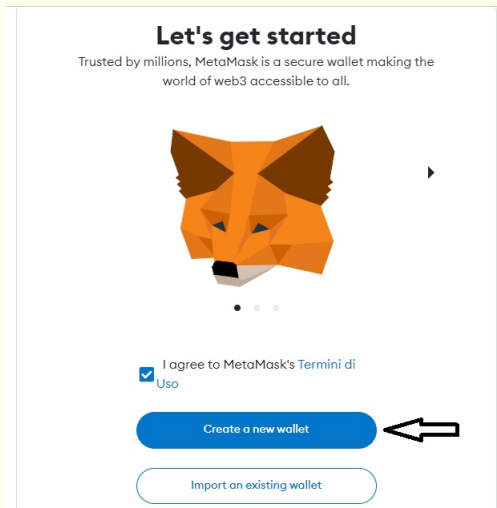


Figure 1: Prima installazione di MetaMask

Write down your Secret Recovery Phrase

Write down this 12-word Secret Recovery Phrase and save it in a place that you trust and only you can access.


Suggerimenti:

- Save in a password manager
- Store in a safe deposit box
- Write down and store in multiple secret places

1. vacant	2. skirt	3. envelope
4. cotton	5. position	6. drift
7. link	8. conduct	9. file
10. gaze	11. work	12. vast

Figure 2: Frase di sicurezza

Frase di sicurezza da completare



1 Crea password 2 Secure wallet 3 Confirm secret recovery phrase

Confirm Secret Recovery Phrase

Confirm Secret Recovery Phrase

1. vacant	2. skirt	3. <input type="text"/>
4. <input type="text"/>	5. position	6. drift
7. link	8. <input type="text"/>	9. file
10. gaze	11. work	12. vast

Figure 3: Frase di sicurezza con parole mancanti

Recupero account

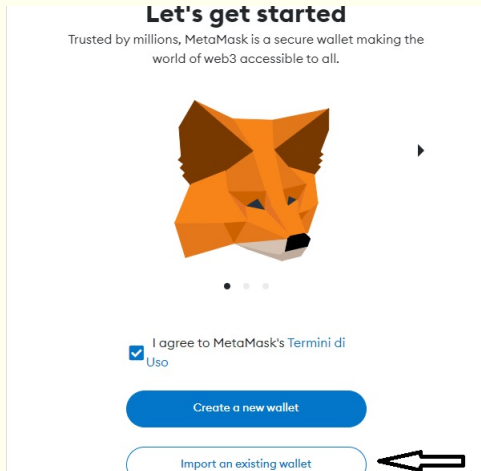


Figure 4: Import di *account* già esistente

Recupero frase seed

Accedi al tuo portafoglio con la tua frase di recupero segreta

MetaMask non può recuperare la tua password. Useremo la tua frase di recupero segreta per assicurarci tu sia il proprietario, ripristinare il tuo portafoglio e impostare una nuova password. Innanzitutto, inserisci la frase di recupero segreta che ti è stata data in fase di creazione del tuo portafoglio. [Scopri di più](#)

Type your Secret Recovery Phrase

I have a 12-word phrase

 You can paste your entire secret recovery phrase into any field

1.	<input type="text"/>		2.	<input type="text"/>		3.	<input type="text"/>	
4.	<input type="text"/>		5.	<input type="text"/>		6.	<input type="text"/>	
7.	<input type="text"/>		8.	<input type="text"/>		9.	<input type="text"/>	
10.	<input type="text"/>		11.	<input type="text"/>		12.	<input type="text"/>	

Confirm Secret Recovery Phrase

Figure 5: Recupero frase seed

Aggiungere un account

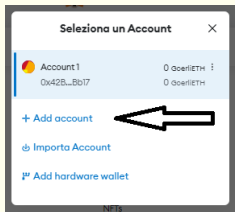


Figure 6: Aggiunta di un proprio *account*

- Gestire i propri *account*;
- Accedere a un *faucet* per ottenere crittovalute sulle *testnet*;
- Inviare transazioni ad altri *address*;
- Funzionalità avanzate che esulano dallo scopo del corso.

Esempio di schermata MetaMask

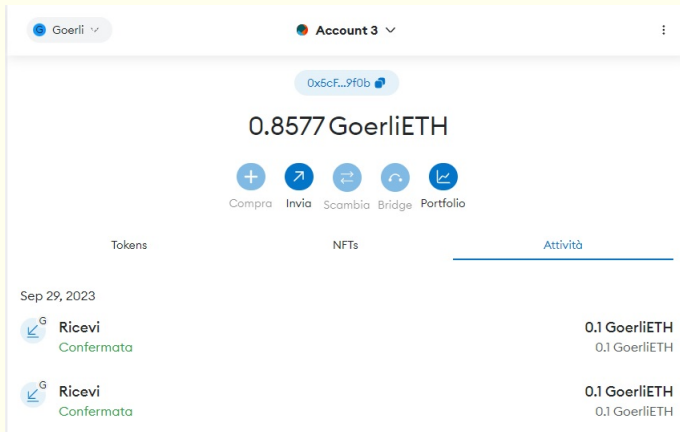


Figure 7: Esempio di schermata

Gestione dei propri account

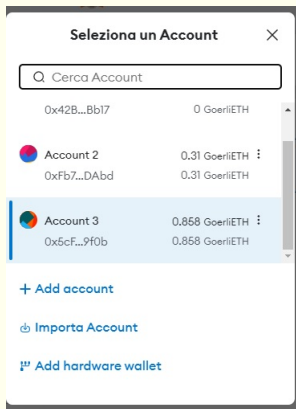


Figure 8: Gestione dei propri *account*

Scelta del network

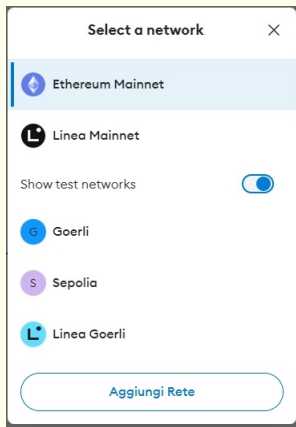


Figure 9: Scelta del network

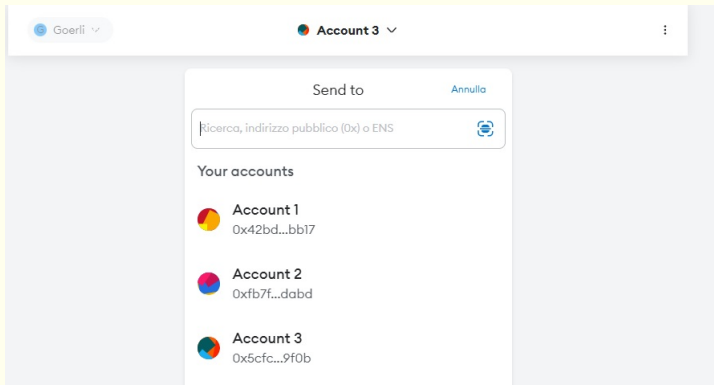


Figure 10: Rubrica

Invio di transazioni

Goerli

Account 3

Invia

Account 1

0x42bd03c8be8ea54bfff7984cfe74167e402bbb17

Asset:

GoerliETH

Bilancio: 0.85770585 GoerliETH

Importo:

Massimo

0.4 GoerliETH

Tasso di conversione non disponibile

Gas (estimated)

0.0000315 GoerliETH

Likely in < 30 seconds

Max fee: 0.0000315 GoerliETH

Figure 11: Invio di transazioni


GOERLI FAUCET

Fast and reliable. 0.02 Goerli ETH/day.

To prevent bots and abuse, the Goerli faucet requires a minimum mainnet balance of 0.001 ETH on the wallet address being used. Join the Ethereum [ecosystem migration](#)! Please use the [Sepolia faucet](#) instead.

Send Me ETH

[Please signup or login](#) with Alchemy to request ETH. It's free!

☐ Non sono un robot
reCAPTCHA
Privacy - Terms

Your Transactions

Time


Figure 12: *Faucet* di Goerli

SEPOLIA FAUCET

Fast and reliable. 0.5 Sepolia ETH/day.

Send Me ETH

✅ Alchemy account connected, receive 0.5 Sepolia ETH! Try out the most effective blockchain sdk - the [Alchemy SDK](#) !

☐ Non sono un robot
reCAPTCHA
Privacy - Terms

Your Transactions	Time
💎 0x35f130017a1e0d5795421072e8c691d559420bc865354308b2d814eaca8a9ad5	23 hours ago

Figure 13: *Faucet* di Sepolia

MetaMask SDK

Nelle prossime *slides* approfondiremo l'integrazione di MetaMask con una propria Dapp scritta in React (basato su JavaScript), uno dei *framework* più utilizzati per realizzare Dapp.

Nelle prossime *slides* approfondiremo l'integrazione di MetaMask con una propria Dapp scritta in React (basato su JavaScript), uno dei *framework* più utilizzati per realizzare Dapp.

Alcuni concetti che vedremo nel seguito:

Nelle prossime *slides* approfondiremo l'integrazione di MetaMask con una propria Dapp scritta in React (basato su JavaScript), uno dei *framework* più utilizzati per realizzare Dapp.

Alcuni concetti che vedremo nel seguito:

- npm
Store di installazione di API per React;

Nelle prossime *slides* approfondiremo l'integrazione di MetaMask con una propria Dapp scritta in React (basato su JavaScript), uno dei *framework* più utilizzati per realizzare Dapp.

Alcuni concetti che vedremo nel seguito:

- `npm`
Store di installazione di API per React;
- `let <name> = <value>`
Istanziamento di una variabile assegnandone il valore;

Prerequisiti

Nelle prossime *slides* approfondiremo l'integrazione di MetaMask con una propria Dapp scritta in React (basato su JavaScript), uno dei *framework* più utilizzati per realizzare Dapp.

Alcuni concetti che vedremo nel seguito:

- `npm`

Store di installazione di API per React;

- `let <name> = <value>`

Istanziamento di una variabile assegnandone il valore;

- `async ... await ...`

Il modificatore `await` consente di attendere l'esecuzione di un comando che sia stato precedentemente dichiarato `async`; `await` può essere sostituito con il metodo `.then()`.

- Nella cartella col proprio progetto installare l'SDK tramite

```
npm i @metamask/sdk
```

- Nello *script* del progetto importare il *provider* di *MetaMask* tramite

```
import { MetaMaskProvider } from '@metamask/sdk-react';
```

Collegamento a un *wallet* tramite MetaMask.

```
const connectWallet = async () => {  
  if(window.ethereum) {  
    const addressArray = await window.ethereum.request({  
      method: 'eth_requestAccounts',  
    })  
  
    let obj = {  
      address: addressArray[0],  
      status: 'Connesso con address: ' + addressArray[0],  
    }  
  
    return obj  
  }  
}
```

Ottenimento del saldo¹ di un *wallet* fornendo in input il suo *address*.

```
1  const updateBalance = (walletAddress) => {
2      window.ethereum.request({
3          method: 'eth_getBalance',
4          params: [walletAddress, 'latest'],
5      }).then( result => {
6          let balance = parseInt(result, 16) / (10**18)
7          setMessage2('Il saldo è di ' + balance + ' <valuta> ether.')
8      })
9  }
```

¹La conversione a riga 6 consente la corretta rappresentazione del saldo.

MetaMask offre ulteriori funzionalità, tra cui eseguire Dapp nel proprio *development network*.

Per chi fosse interessato a questa e ad altre funzionalità suggeriamo di consultare la documentazione dal sito ufficiale <https://metamask.io/>

Solidity

Solidity è il più famoso e conosciuto linguaggio per scrivere *smart contract*.

Solidity è il più famoso e conosciuto linguaggio per scrivere *smart contract*.

Osservazione

Gli *smart contract* rappresentano il *backend* di un'applicazione decentralizzata.

Solidity è il più famoso e conosciuto linguaggio per scrivere *smart contract*.

Osservazione

Gli *smart contract* rappresentano il *backend* di un'applicazione decentralizzata.

Vedremo direttamente alcuni esempi pratici che ci mostreranno le potenzialità del linguaggio.

L'architettura di una Dapp può essere suddivisa in *backend* e *frontend*.

L'architettura di una Dapp può essere suddivisa in *backend* e *frontend*.

- La parte di *frontend* è quella riguardante l'interfaccia utente e non differisce dalle applicazioni tradizionali;

L'architettura di una Dapp può essere suddivisa in *backend* e *frontend*.

- La parte di *frontend* è quella riguardante l'interfaccia utente e non differisce dalle applicazioni tradizionali;
- La parte di backend viene solitamente gestita dagli *smart contract*.

- `string nome = "Christopher";`
- `int temperatura = -6;`
- `uint giorniAlNatale = 25;`
- `bool b = false;`
- `address public account = 0x5A0b54D...E029c4c;`
- `int[] public nomi = [0, 1, 2, 3, 4];`

- Operazioni aritmetiche:
 - Somma: $c = a + b$;
 - Sottrazione: $c = a - b$;
 - Prodotto: $c = a * b$;
 - Quoziente: $c = a / b$;
 - Modulo: $c = a \% b$.

- Operazioni aritmetiche:
 - Somma: $c = a + b$;
 - Sottrazione: $c = a - b$;
 - Prodotto: $c = a * b$;
 - Quoziente: $c = a / b$;
 - Modulo: $c = a \% b$.
- Operazioni logiche:
 - Uguaglianza: $x == y$;
 - Diversità: $x != y$;
 - Disuguaglianze strette: $x < y$ e $x > y$;
 - Disuguaglianze: $x <= y$ e $x >= y$;

Prime parole chiave

public: Indica una variabile o una funzione visibile a chiunque;

view: Indica una funzione può leggere, ma non modificare, le variabili di stato dello *smart contract*;

returns: Si usa al termine definizione di una funzione per indicare il tipo di variabile che la funzione produce in *output*;

return: Si usa al termine delle istruzioni di una funzione per indicare l'*output* da restituire.

```
function <functionName>(<params>) public view returns (<outputType>) {  
    ...  
  
    return <outputVariable>;  
}
```

- Le funzioni sono *script* di codice contenente delle istruzioni che verranno eseguite più volte.
- Aiutano la leggibilità dello *smart contract*.

```
function triplica(uint number) public view returns uint {  
    return number * 3;  
}
```

- La struct si utilizza quando si vuole definire un'entità che rappresenta un oggetto del mondo reale

```
struct Bottiglia {  
    string name;  
    string image;  
    uint year;  
}
```

```
Bottiglia[] bottiglie;
```

```
function setB(string memory name, string memory image, uint year) public {  
    Bottiglia memory bottiglia = Bottiglia(name, image, year);  
    bottiglie.push(bottiglia);  
}
```

If clauses

- Il costrutto `if`, `else if`, `else` riprende la logica degli *smart contract*.
- La condizione da verificare può essere composta attraverso gli operatori booleani.
 - `and (&&)`;
 - `or (||)`;
 - `not (!)`.

```
function isOdd(int number) public view returns (string memory) {  
    if (number%2 == 1 && number > 0)  
        return "Il numero è dispari";  
    else if (number%2 == 0 && number >= 0)  
        return "Il numero NON è dispari";  
    else  
        return "Ci interessano soltanto i numeri non negativi";  
}
```

While loop

- Il costrutto `while` si usa quando è necessario iterare delle istruzioni finché una certa condizione rimane vera.
- Se la condizione è falsa in partenza, le istruzioni non vengono mai eseguite.

```
function incrementa(uint number) public view returns (uint) {  
    counter = 0;  
    i = 1;  
  
    while (counter < 16) {  
        i = i * 2;  
        counter++;  
    }  
  
    return i;  
}
```

For loop

- I cicli for si possono usare in alternativa al while per avere una scrittura più compatta.

```
function inc(uint[] memory num) public view returns (uint[] memory) {
    uint len = num.length;

    for (uint i = 0; i < len; i++) {
        num[i] = num[i] + i;
    }

    return num;
}
```

Altre parole chiave i

`payable`: Si usa per tutte le funzioni, gli *address* o i contratti che si aspettano di ricevere ether;

`msg.sender`: Indica l'*address* di colui che sta chiamando lo *smart contract*;

`msg.value`: Si usa solitamente al momento di eseguire transazioni per indicare quanti soldi devono essere trasferiti.

`constructor`: Funzione che viene chiamata soltanto una volta, al momento del *deploy*; non è possibile per qualcuno che usa lo *smart contract* chiamare questa funzione.

```
address proprietario;
```

```
constructor(string memory input) public{  
    proprietario = msg.sender;  
}
```

Require: La parola chiave pone un requisito a cui è vincolata l'esecuzione della funzione.

```
contract ModificaTemperaturaRilevata{
    int t = -10;
    address ammiraglio = 0xFb7F25BD8cc61Fa9B32Dc8a79c6C5944FeafDAbd;

    function aggiornaTemperatura(int newT) {
        require(msg.sender == ammiraglio);
        t = newT;
    }
}
```

`event` ed `emit` sono due parole chiave necessarie per l'interazione di un'applicazione decentralizzata con lo *smart contract* e le vedremo attraverso un esempio pratico.

Intuitivamente, quando un evento (`event`) viene emesso (`emit`), salva sulla *blockchain* gli argomenti passati attraverso la transazione.

Si può accedere a questi dati conoscendo l'*address* dello *smart contract*.

Come realizzare uno *smart contract*? Ci viene in aiuto un apposito ambiente di sviluppo.

Come realizzare uno *smart contract*? Ci viene in aiuto un apposito ambiente di sviluppo.

- Si chiama Remix.

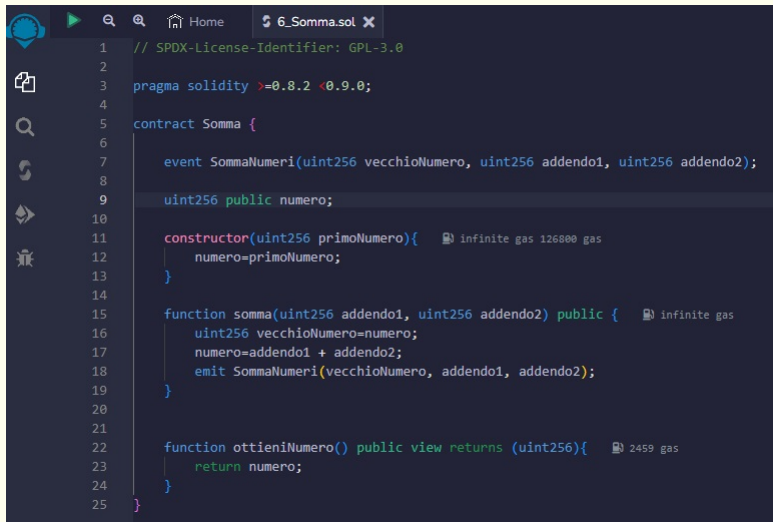
Come realizzare uno *smart contract*? Ci viene in aiuto un apposito ambiente di sviluppo.

- Si chiama Remix.
- Utilizza Solidity.

Un esempio di smart contract

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.2 <0.9.0;
contract Somma {
    event SommaNumeri(uint256 vecchioNumero,uint256 addendo1,uint256 addendo2);
    uint256 public numero;
    constructor(uint256 primoNumero){
        numero=primoNumero;
    }
    function somma(uint256 addendo1, uint256 addendo2) public {
        uint256 vecchioNumero=numero;
        numero=addendo1 + addendo2;
        emit SommaNumeri(vecchioNumero, addendo1, addendo2);
    }
    function ottieniNumero() public view returns (uint256){
        return numero;
    }
}
```

Remix - Esempio smart contract



The image shows the Remix IDE interface with a Solidity smart contract named 'Somma' open in the editor. The contract is written in Solidity and includes a constructor, a public function 'somma', and a public view function 'ottieniNumero'. The code is as follows:

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract Somma {
6
7     event SommaNumeri(uint256 vecchioNumero, uint256 addendo1, uint256 addendo2);
8
9     uint256 public numero;
10
11     constructor(uint256 primoNumero){ infinite gas 126800 gas
12         numero=primoNumero;
13     }
14
15     function somma(uint256 addendo1, uint256 addendo2) public { infinite gas
16         uint256 vecchioNumero=numero;
17         numero=addendo1 + addendo2;
18         emit SommaNumeri(vecchioNumero, addendo1, addendo2);
19     }
20
21
22     function ottieniNumero() public view returns (uint256){ 2459 gas
23         return numero;
24     }
25 }
```

Figure 14: Esempio di *smart contract* su Remix

Remix - Compiler

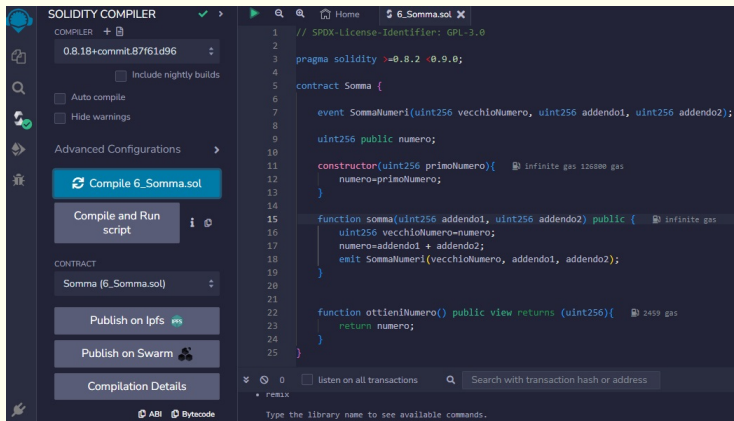


Figure 15: Compiler di Remix

Remix - Deploy

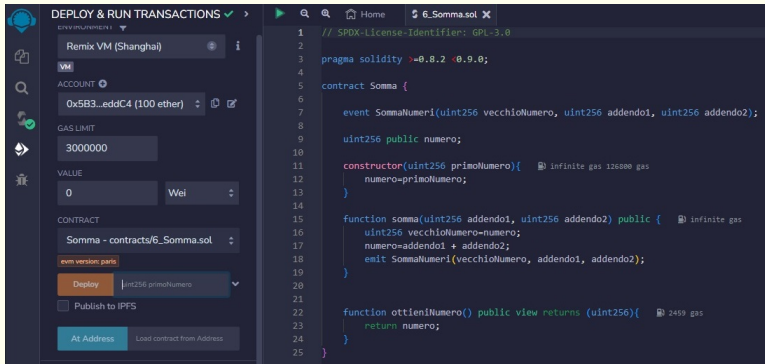


Figure 16: Deploy di *smart contract* su remix

Remix - Chiamata di funzioni

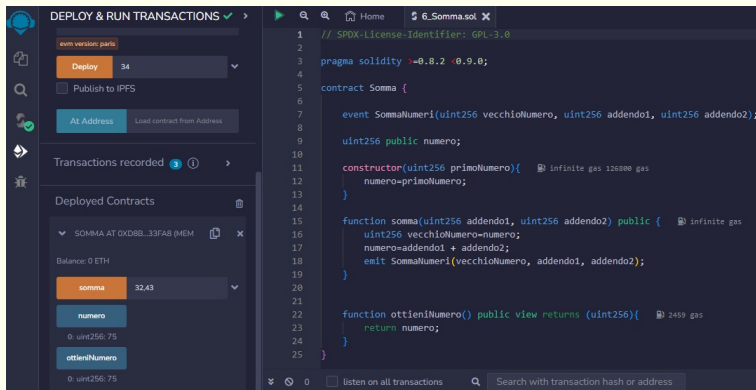


Figure 17: Chiamata di funzioni dello *smart contract*

Remix - Environment

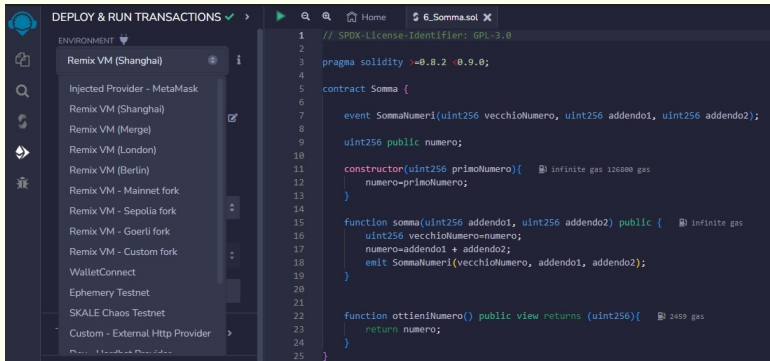


Figure 18: Scelta dell'environment

Connettere una dapp a uno smart contract

- Attraverso l'applicazione decentralizzata è possibile chiamare le varie funzioni scritte sullo *smart contract*.
- Per farlo è necessario definire un **host**, l'**address** dello *smart contract* e il suo **ABI**

```
let host=window.ethereum;
let sca="0x2AbCaBBe46958191a1d63EC8C5801C1490cC7839";
let cABI=[
  {
    "inputs": [
      {
        "internalType": "uint256",
        //...
      },
    ],
    "stateMutability": "view",
    "type": "function"
  }
];
```

- Attraverso la Dapp è possibile richiamare una funzione di uno *smart contract* che salva valori sulla *blockchain*.

```
async function setSomma(host, CABI, sca,a1,a2){  
  
  const web3 = new Web3(host);  
  const c = new web3.eth.Contract(cABI, sca);  
  //Il .send è necessario perché sto interagendo  
  //con lo smartcontract attivamente  
  const res;  
  res=await c.methods.somma(a1,a2).send({  
    from:w,gasPrice:"0xFF",  
    gasLimit:"0xFFFFFFFF"  
  });  
  return res;  
}
```

- Attraverso l'applicazione decentralizzata è possibile anche richiamare una funzione di uno *smart contract* che legge valori salvati sulla *blockchain*.

```
async function getSomma(host,cABI,sca){  
    const web3 = new Web3(host);  
    const c = new web3.eth.Contract(cABI, sca);  
    const res = await c.methods.ottieniNumero().call();  
    setSommaSuBlockchain(res);  
}
```

Eseguire una transazione

- Oltre all'interfaccia di MetaMask, anche una dapp può essere utilizzata per inviare transazioni.

```
const onTransactionPressed = async () => {  
  
  let t={  
    to: targetAddress,  
    from: walletAddress,  
    value: (valoreDaInviare*(10**18)).toString(16),  
  };  
  window.ethereum.request({method:"eth_sendTransaction",params:[t]}).then(  
    txhash=>{hashTransazione=txhash;})  
}
```

Dapp - Prima schermata dapp

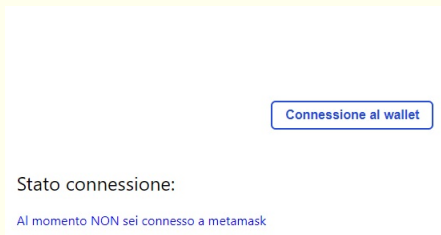


Figure 19: Schermata di apertura della dapp



Figure 20: Bottone per richiedere il proprio bilancio

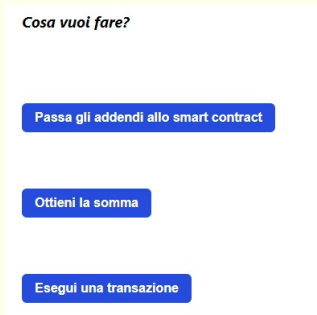


Figure 21: Menu della Dapp

Invia due addendi allo smart contract
0x2AbCaBBE46958191a1d63EC8C5801C1490cC7839

Addendo numero 1

Addendo numero 2

Esegui la somma

Figure 22: Funzionalità di somma



Ottieni saldo aggiornato

Il tuo saldo aggiornato è di: 0.310298384168772 goerli ether

Valore da inviare

AddressRicevente

Esegui la transazione

Figure 23: Transazioni tramite Dapp

Grazie per l'attenzione