

UNIVERSITA' DEGLI STUDI DI NAPOLI
“PARTHENOPE”

- Corso di Laurea in Informatica –



Reti di Calcolatori e Laboratorio Reti di Calcolatori - A.A.
2019/2020

Prof. Alessio Ferone

Progetto esame:

Event Management

Studente: Simone Fuso

Matricola: 0124001026

Indice

Indice.....	2
1 Descrizione del progetto.....	4
1.1 Traccia del progetto.....	4
2 Una panoramica	5
2.1 Blockchain.....	5
2.1.1 Interagire con la blockchain: wallet	6
2.2 Ethereum: the world computer.....	6
2.2.1 Ethereum: account-based blockchain	7
2.2.2 Tansazioni	7
2.2.3 DApp.....	8
3 Descrizione e schemi dell'architettura	9
3.1 Descrizione.....	9
3.1.1 Schema dell'architettura	9
4 Descrizione e schemi del protocollo applicazione	10
4.1 Descrizione del protocollo applicazione	10
4.2 Schemi del protocollo applicazione	10
5 Dettagli implementativi.....	12
5.1 Solidity e EVM	12
5.1.1 Contratto A	12
5.1.2 Contratto B	13
5.1.3 Migrazioni	14
5.2 Interfaccia Web	15

6	Manuale Utente.....	17
6.1	Requisiti	17
6.2	Settaggi generali.....	17
6.3	Home Page	20

1 Descrizione del progetto

1.1 Traccia del progetto

Si vuole realizzare un sistema di prenotazione di eventi basato sulla blockchain di Ethereum con l'uso di smart contracts.

L'utente registra la propria presenza ad un evento, inviando la quantità di ether richiesta ad un contratto. Successivamente, l'utente che si è registrato all'evento, effettua il check-in richiamando il contratto B, che restituisce all'utente la quantità di ether inviata in fase di registrazione. Se l'utente non effettua il check-in entro un tempo massimo stabilito dall'evento, il contratto trattiene gli ether precedentemente inviati. Alla scadenza del tempo massimo, tutti gli ether non restituiti possono essere riscossi dal proprietario del contratto.

Il sistema si compone dei seguenti contratti:

Contratto A:

- verifica che la quantità di ether inviati è corretta
- verifica che l'indirizzo dell'utente non sia già registrato
- registra l'indirizzo dell'utente
- invia gli ether al contratto B

• Contratto B:

- verifica se l'indirizzo dell'utente che sta eseguendo il check-in è registrato nel contratto A
- in caso affermativo gli restituisce gli ether inviati
- alla scadenza del tempo massimo, trasferisce al proprietario del contratto gli ether non restituiti

2 Una panoramica

In questa prima sezione faremo una breve panoramica sulla blockchain e la piattaforma Ethereum.

2.1 Blockchain

La blockchain è una sottofamiglia di tecnologie in cui il registro è strutturato come una catena di blocchi contenenti le transazioni (vedi fig. 2.1) e la cui validazione è affidata a un **meccanismo di consenso** distribuito su tutti i nodi che partecipano alla rete. Le principali caratteristiche delle tecnologie blockchain sono l'immutabilità del registro, la trasparenza, tracciabilità delle transazioni e la sicurezza basata su tecniche crittografiche. La blockchain è basata su una rete p2p e dal punto di vista delle funzionalità permette di gestire un database in modo distribuito. Dal punto di vista operativo è una alternativa agli archivi centralizzati e permette di gestire l'aggiornamento dei dati con la collaborazione dei partecipanti alla rete, con la possibilità di avere dati condivisi, accessibili e distribuiti presso tutti i partecipanti. Di fatto, permette una gestione dei dati in termini di verifica e di autorizzazione senza che sia necessaria una autorità centrale.

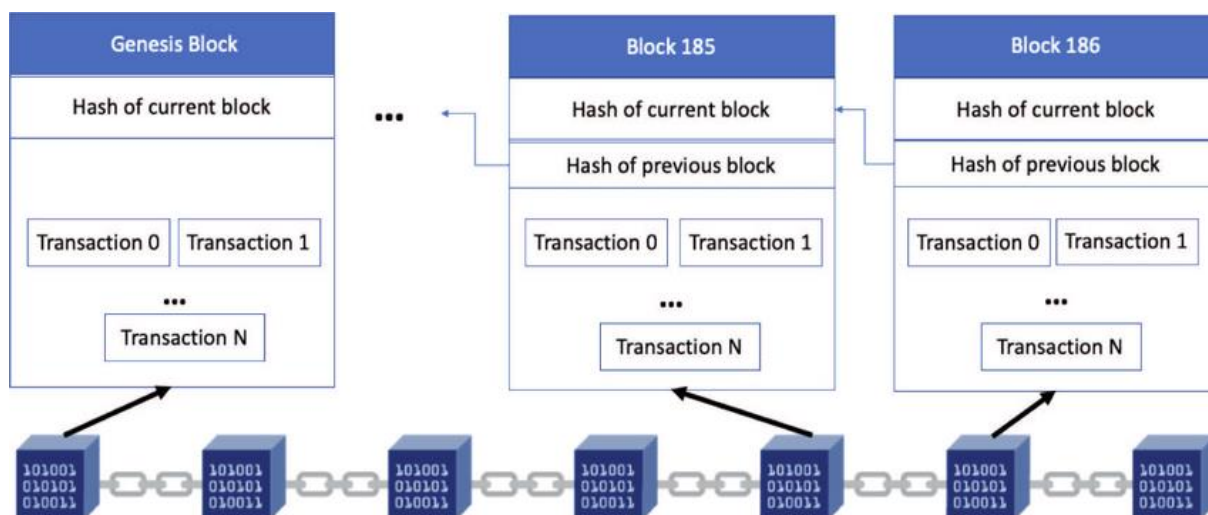


Figura 2.1: Schema di una catena di blocchi di una blockchain

2.1.1 Interagire con la blockchain: wallet

Un wallet è un software usato per interagire con la blockchain. Gestisce uno o più indirizzi ad ognuno dei quali sono associate delle chiavi, ottenute, mediante crittografia a chiave pubblica. Il wallet tiene traccia del bilancio, crea e firma le transazioni (con la chiave privata corrispondente all'indirizzo).

2.2 Ethereum: the world computer

Ethereum(vedi fig. 2.2) è una state machine deterministica ma praticamente illimitata, costituita da un **singolo stato** globalmente accessibile ed una **virtual machine** che modifica questo stato. Informalmente, si tratta di un'infrastruttura software distribuita che esegue programmi chiamati **smart contract** ed utilizza una blockchain per sincronizzare e memorizzare lo stato condiviso. In Ethereum si utilizza una criptomoneta chiamata ether.

Ethereum è dotato di un linguaggio **Turing completo** in grado di eseguire un programma memorizzato nella blockchain tramite una virtual machine. Ovvero può funzionare come un computer general purpose, questo è il motivo per cui viene chiamato “the world computer”.

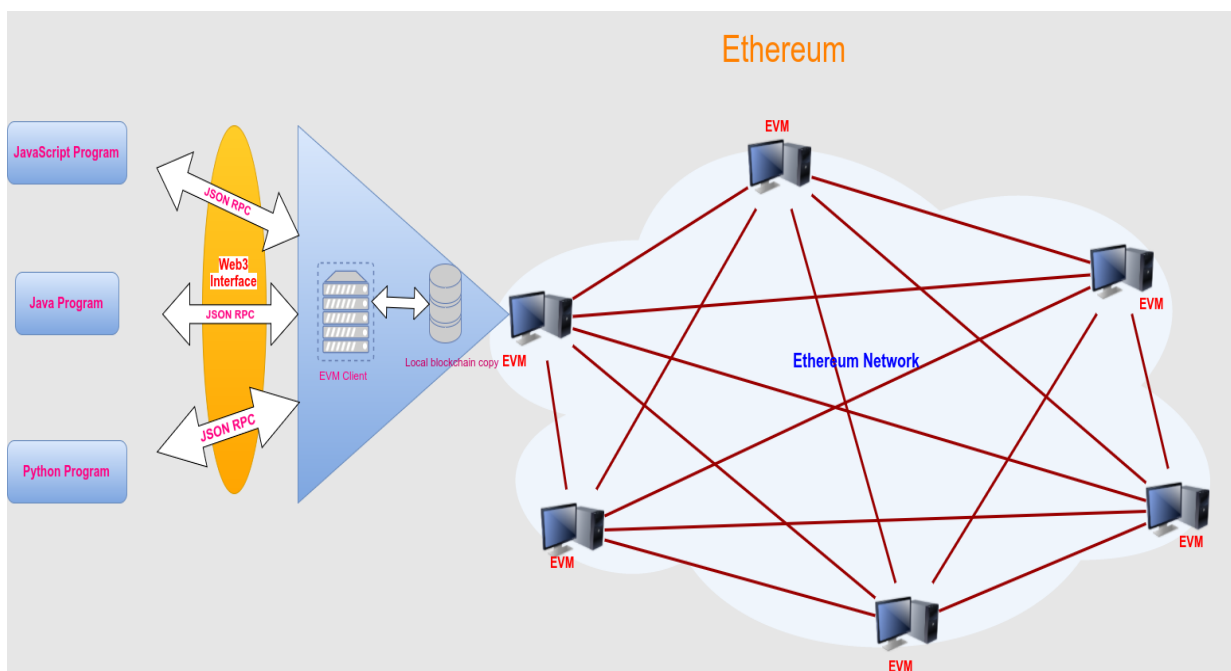


Figura 2.2: Schema di massima della struttura Ethereum.

Per molti aspetti è molto simile a Bitcoin, ma ha delle caratteristiche uniche. Bitcoin può essere considerata come una state machine basata sul consenso distribuito in cui le transazioni determinano la **transizione dello stato globale** modificando la proprietà delle monete virtuali. Ethereum oltre a fare ciò, memorizza anche lo stato di un **data-store general-purpose**, ovvero una memoria in grado di memorizzare ogni dato esprimibile come coppia chiave-valore. Questa memoria funziona come una RAM e la blockchain è utilizzata per determinare i cambiamenti di stato. Rispetto ad un computer general-purpose, i cambiamenti di stato sono governati dalle **regole del consenso** (protocollo) e lo **stato** è **distribuito** globalmente. Per determinare la terminazione di un programma, ogni istruzione eseguita dalla EVM ha un costo in unità chiama **gas**. Quando una transazione richiede l'esecuzione di uno smart contract deve includere una quantità di gas che è l'upper bound per l'esecuzione dello smart contract. Terminato il gas messo a disposizione l'esecuzione viene terminata. Il gas viene acquistato mediante gli ether.

2.2.1 Ethereum: account-based blockchain

Ethereum è un implementazione di blockchain account-based. Esistono due tipi di account in Ethereum: Externally owned account è un tipo di account dotato di chiave privata che permette il controllo dell'account. Un altro tipo di account è il contract account che non ha una chiave privata e sono controllati dalla logica dello smart contract a cui sono associati.

2.2.2 Tansazioni

I contratti possono reagire ad una transazione chiamando altri contratti o creando transazioni per trasferire ether. Le transazioni sono messaggi firmati da un EOA, trasmesse dalla rete Ethereum e memorizzate su blockchain, implicano un cambiamento di stato o l'esecuzione di un contratto. Prima di essere trasmessa una transazione deve essere serializzata e contiene i seguenti campi:

- **nonce**: un numero sequenziale, generato da un EOA
- **gasPrice**: prezzo del gas che si è disposti a pagare
- **gasLimit**: massima quantità di gas che si vuole comprare/utilizzare
- **recipient**: indirizzo del destinatario
- **value**: quantità di ether da inviare

- *data*: dati informato binario
- *v, r, s*: parametri della firma digitale

2.2.3 DApp

Ethereum (vedi fig 2.2) si è evoluto fino ad arrivare ad essere una piattaforma per programmare **DApp** (Decentralized App), che è costituita da uno smart contract e un'interfaccia web. Si tratta di una web app costruita al di sopra della blockchain.

3 Descrizione e schemi dell'architettura

3.1 Descrizione

Event Management è una DApp (vedi sez. [2.2.3](#)). Fornisce tutte le informazioni all'utente affinché possa effettuare le transazioni necessarie. Ovvero è possibile creare un evento, registrare la presenza dell'utente, ed effettuare il check-in all'evento.

3.1.1 Schema dell'architettura

Ci sono due tipi di utilizzatori del servizio Event Management (vedi fig. 3.1). Da un lato c'è il proprietario dei contratti, che è colui che distribuisce sulla blockchain Ethereum gli smart contract. Può creare un evento e riscuotere gli ether degli utenti che non hanno effettuato il check-in, nel tempo massimo stabilito. Dall'altro lato, c'è l'utente che usufruisce del servizio. Può registrarsi all'evento ed effettuare il check-in, entro il tempo massimo stabilito dall'evento. Per permettere agli attori di usare i servizi messi a disposizione, sono stati implementati due smart contract, che possono interagire tra loro o con gli utenti mediante transazioni.

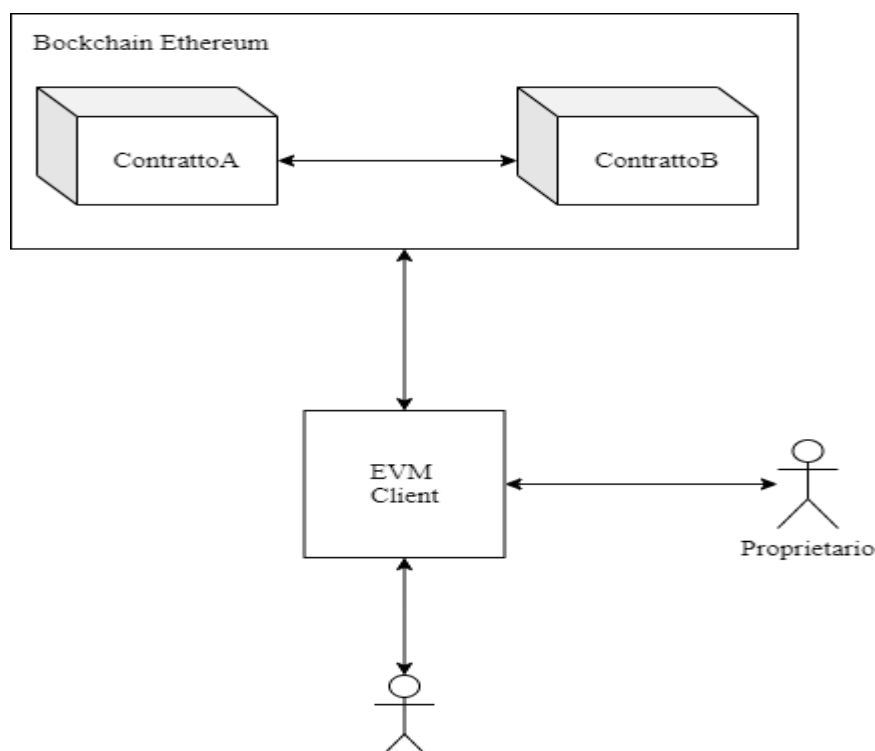


Figura 3.1: Schema dell'architettura della DApp Event Management.

4 Descrizione e schemi del protocollo applicazione

4.1 Descrizione del protocollo applicazione

Il protocollo applicazione deve implementare la logica dei contratti, per far sì, che il proprietario possa creare l'evento e riscuotere gli ether non inviati. Mentre l'utente si possa registrare, effettuare il check-in e ricevere gli ether precedentemente inviati. Inoltre deve fornire un metodo per far comunicare i contratti tra loro, un utente con un contratto e viceversa.

Vedremo in seguito come avviene lo scambio dei messaggi tra le parti interessate.

4.2 Schemi del protocollo applicazione

All'interno della blockchain Ethereum, lo scambio di messaggi avviene tramite le transazioni (vedi sez. [2.2.2](#)). Gli attori inviano delle transazioni con tutte le informazioni necessarie per interagire con i contratti (vedi fig. [3.1](#)). In particolare il protocollo applicazione prevede un scambio iniziale di messaggi mediante una transazione tra l'account del proprietario, che intende creare un evento, e il "Contratto A". In questa transazione, iniziale, il proprietario specifica, il nome dell'evento, la quantità di ether necessaria per effettuare la registrazione all'evento e il tempo massimo per effettuare il check-in (espresso in secondi). Sarà compito di tale contratto verificare la correttezza dei dati ricevuti e registrare tutte le informazioni nell'apposita blockchain. Mediante il campo mittente della transazione, il contratto è in grado di verificare se l'indirizzo dell'account che ha fatto richiesta di creazione dell'evento sia quello del proprietario.

Creato l'evento, l'utente che intende registrarsi all'evento, invia una transazione al "Contratto A" con la quantità di ether richiesta. E' compito del Contratto A controllare, la quantità di ether inviati, che il tempo per effettuare il check-in non sia scaduto e salvare i campi come l'indirizzo dell'utente e a quale evento si è registrato. Infine invia gli ether ricevuti al Contratto B che li terrà fino al check-in da parte dell'utente o dal riscatto da parte del proprietario. A quel punto gli utenti che avranno effettuato il check-in, avviando una transazione verso il Contratto B, vedranno restituiti gli ether in cauzione.

Nel caso in cui il proprietario volesse riscattare tutti gli ether, invia una transazione al contratto B, che a sua volta, controlla che il tempo massimo per effettuare il check-in sia scaduto. In caso affermativo, il contratto B, invia una

transazione contenente, gli ether che non sono stati riscattati al proprietario. Di seguito è mostrato lo schema del protocollo applicazione.

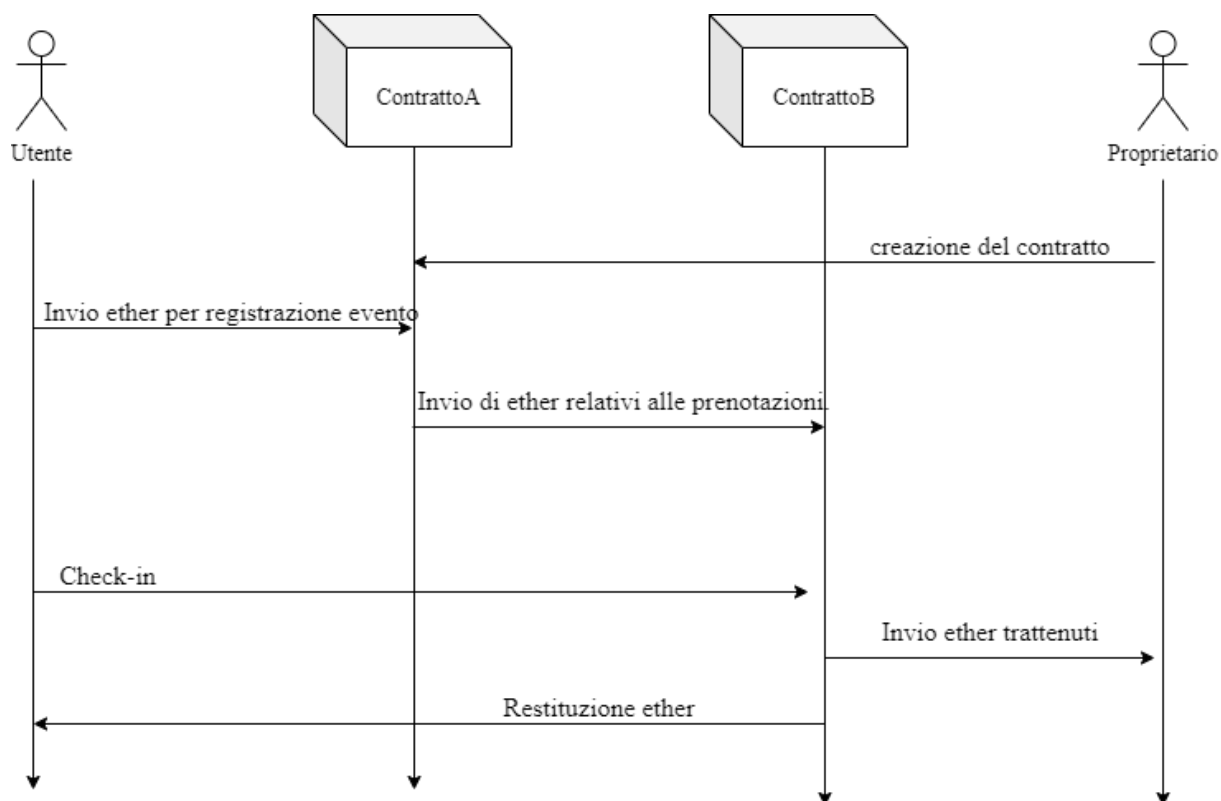


Figura 4.1: Schema del protocollo applicazione

5 Dettagli implementativi

5.1 Solidity e EVM

Gli smart contracts sono scritti in Solidity , un linguaggio ad alto livello orientato agli oggetti. È stato influenzato dal C++, Python e Javascript. Compilare un file .sol significa tradurre ciò che è scritto in Solidity in bytecode eseguibile dalla EVM (Ethereum Virtual Machine), in maniera molto simile a ciò che accade con Java. Eseguire operazioni con la EVM cambia lo stato globale di Ethereum (essendo lo stato condiviso, in quanto blockchain). Inoltre, ogni operazione eseguita costa del gas, ovvero del "carburante" acquistato con la valuta della blockchain. In questo modo è scongiurata la possibilità del **Halting problem** e che quindi possa bloccare l'intero sistema. Possibili esecuzioni di codice malevolo atto a bloccare il sistema si ridurranno al solo spreco di gas da parte dell'esecutore dell' "attacco"; ne consegue che anche il tentativo di bloccare per un tempo considerevole il sistema si traduce, in un' attività economicamente dispendiosa.

5.1.1 Contratto A

In questa sezione vediamo alcune funzioni del Contratto A.

La porzione di codice in basso effettua la creazione dell'evento. E' una funzione pubblica. Ha un modificatore di funzione **onlyOwner**, ovvero la funzione può essere chiamata solo dal proprietario del contratto. Incrementa il contatore degli eventi e assegna, alla mapping(uint => Event) events, la struttura Event con i vari campi che la compongono passati in input.

```
51 // il proprietario del contratto è l'unico che può creare l'evento
52 function createEvent(string memory _nameEvent, uint _amount, uint _timeSecondCheckIn) public onlyOwner {
53     eventCount++;
54     events[eventCount] = Event(eventCount, _nameEvent, _amount, now + _timeSecondCheckIn, 0);
55 }
```

La funzione **register** permette ad un utente di registrarsi all'evento. E' una funzione pubblica e payable, il che significa che quest'ultima può ricevere Ether. In input prende l'id dell'evento. **checkAmount**, controlla che la quantità di ether inviata sia corretta. Successivamente **require** controlla che l'utente non sia registrato a quell'evento. In caso affermativo imposta a true il campo **_isregister**, così che l'utente non si possa registrare più volte allo stesso evento. Aggiorna il bilancio dell'evento e trasferisce gli ether inviati al Contratto B.

```
63 // verifica che l'indirizzo dell'utente non sia registrato in caso affermativo lo registra e invia gli ether al contratto B
64 function register(uint _idEvent)public payable {
65     checkAmount(_idEvent);
66     require(!users[msg.sender][_idEvent]._isregistered); //verifica che l'utente non sia registrato
67     users[msg.sender][_idEvent]._isregistered=true;
68     events[_idEvent].balance += msg.value; // aggiorna il bilancio
69     contrattob.transfer(msg.value); // trasferisce gli ether ricevuti al contrattoB
70     emit transferToB(msg.sender, contrattob, eventCount, msg.value); // emette l'evento del trasferimento di ether al contratto
71 }
```

5.1.2 Contratto B

Il costruttore viene chiamato quando si effettua il deploy (inviare il contratto alla blockchain) del contratto. Imposta l'indirizzo del proprietario del contratto, l'indirizzo del contratto A e invia a quest'ultimo l'indirizzo del Contratto B.

```
10 constructor(ContrattoA _contrattoa)public{
11     owner = msg.sender;
12     contrattoa = _contrattoa;
13     contrattoa.setAddr(addrB); // invia l'indirizzo del contrattoB al contrattoA
14 }
```

L'utente quando effettua il check-in. In input prende l'id dell'evento. E' external ovvero la funzione può essere chiamata attraverso una transazione da account EOA o da un altro contratto.

```
28 //verifica se l'indirizzo dell'utente che sta eseguendo il check-in è registrato nel contratto A in caso affermativo restituisce gli ether
29 ✓ function checkIn( uint _idEvent) external payable {
30     contrattoA.isRegistered(msg.sender,_idEvent); //verifico se l'indirizzo dell'utente è registrato nel contratto A
31     uint timeCheckIn= contrattoA.getTimeCheckIn(_idEvent);
32     uint amount =contrattoA.getAmount(_idEvent);
33     require(timeCheckIn >= now); //verifico se il tempo per il checkIn è scaduto
34     msg.sender.transfer(amount); //affermativo trasferisce all'utente gli ether precedentemente inviati
35     contrattoA.setBalance(_idEvent,amount,msg.sender); // aggiorna il bilancio dell'evento
36     emit transferAmount(address(this), msg.sender, _idEvent,amount);
37 }
```

La prima riga chiama una funzione nel Contratto A, che verifica se l'indirizzo chiamante è registrato nel contratto A e quindi può effettuare il check-in.

Controlla se l'utente è ancora in tempo per effettuare il check-in, in caso affermativo invia la quantità di ether inviata precedentemente nella registrazione all'evento. In fine imposta il bilancio dell'evento.

transferBalance trasferisce al proprietario, il quale è l'unico che può chiamare la funzione, gli ether non restituiti agli utenti che non hanno effettuato il check-in nel tempo massimo stabilito.

```
39 /*alla scadenza del tempo max per effettuare il check-in, solo il proprietario
40 può chiamare questa funzione trasferire il bilancio dell'evento nel suo wallet */
41 function transferBalance(uint _idEvent) external payable onlyOwner{
42     uint timeCheckIn= contrattoA.getTimeCheckIn(_idEvent);
43     uint balance=contrattoA.getBalance(_idEvent);
44     require(timeCheckIn < now);
45     owner.transfer(balance);
46 }
```

5.1.3 Migrazioni

Una migrazione serve a rilasciare i contratti sulla blockchain. Questa è la porzione di codice che rilascia il Contratto A e B sulla blockchain.

```
const ContrattoA = artifacts.require("ContrattoA");
const ContrattoB = artifacts.require("ContrattoB");
```

```

module.exports = async function(deployer) {
  // Deploy ContrattoA
  await deployer.deploy(ContrattoA)
  const contrattoA = await ContrattoA.deployed()
  // Deploy ContrattoB
  await deployer.deploy(ContrattoB, contrattoA.address)
  const contrattoB = await ContrattoB.deployed()
};

```

5.2 Interfaccia Web

L'interfaccia web è stata implementata con react, una libreria javascript per creare UI. Il codice in calce contiene l'implementazione del collegamento a **Ganache**, ovvero, una blockchain Ethereum locale, per effettuare test. Il codice implementa l'interfaccia ai contratti tramite Web3 e Metamask (wallet).

```

async loadBlockchainData() {
  const web3 = window.web3

  const accounts = await web3.eth.getAccounts()
  this.setState({
    account: accounts[0]
  })

  // Load ContrattoA
  const networkId = await web3.eth.net.getId()
  const contrattoAData = ContrattoA.networks[networkId]
  if(contrattoAData) {
    const contrattoa = new web3.eth.Contract(ContrattoA.abi, contrattoAData.address)
    this.setState({ contrattoa })
  } else {
    window.alert('ContrattoA contract not deployed to detected network.')
  }

  // Load contrattoB
  const contrattoBData = ContrattoB.networks[networkId]
  if(contrattoBData) {
    const contrattob = new web3.eth.Contract(ContrattoB.abi, contrattoBData.address)
    this.setState({ contrattob })
  } else {
    window.alert('ContrattoB contract not deployed to detected network.')
  }
}

```

```

    this.setState({ loading: false })
  }

  async loadWeb3() {
    if (window.ethereum) {
      window.web3 = new Web3(window.ethereum)
      await window.ethereum.enable()
    }
    else if (window.web3) {
      window.web3 = new Web3(window.web3.currentProvider)
    }
    else {
      window.alert('Non-
Ethereum browser detected. You should consider trying MetaMask!')
    }
  }

  createEvent = (name,amount,timeCheckIn) => {
    this.setState({ loading: true })
    this.state.contrattoa.methods.createEvent(name,amount,timeCheckIn).send({
from: this.state.account }).on('transactionHash', (hash) => {
    this.setState({ loading: false })
  })
  }

  register = (Id,amount) => {
    this.setState({ loading: true })
    this.state.contrattoa.methods.register(Id).send({ value:amount , from:
this.state.account }).on('transactionHash', (hash) => {
    this.setState({ loading: false })
  })
  }

  checkIn = (Id) => {
    this.setState({ loading: true })
    this.state.contrattob.methods.checkIn(Id).send({ from: this.state.account
t }).on('transactionHash', (hash) => {
    this.setState({ loading: false })
  })
  }

```

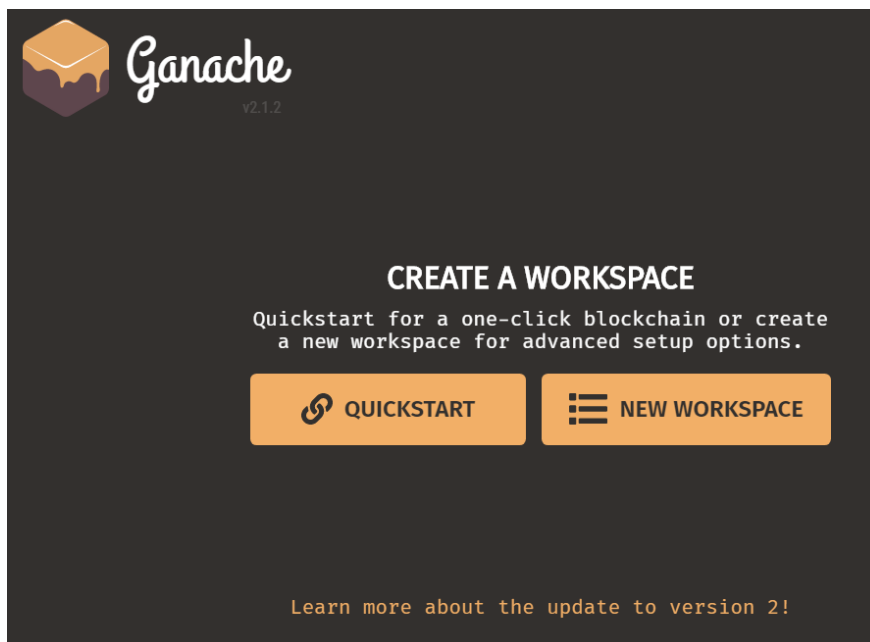

6 Manuale Utente

6.1 Requisiti

- Nodejs <https://nodejs.org/it/>
- Ganache <https://www.trufflesuite.com/ganache>
- Metamask <https://metamask.io/download.html> e relativo account
- Codice <https://github.com/simonefuso/Event-Management>

6.2 Settaggi generali

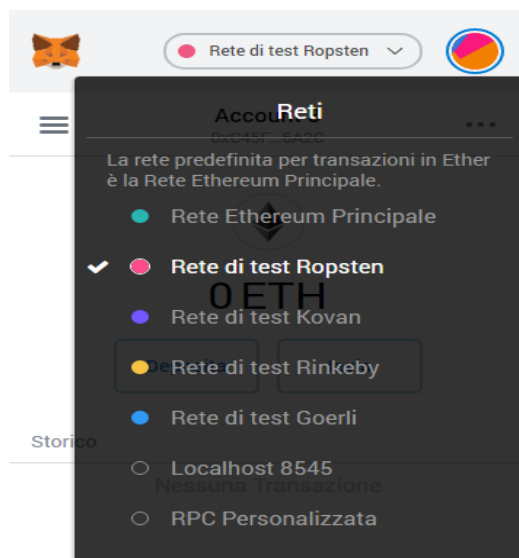
Aprire Ganache e cliccare su QuickStart



Aprire la shell o il prompt dei comandi nella directory principale del progetto:

- Digitare `npm install`
- Digitare `npm install truffle -g`
- Digitare `truffle migrate`
- Digitare `npm run start`

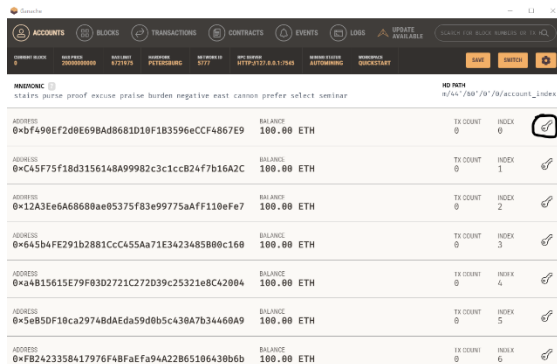
Aprire Metamask ed accedere. Cliccare su Reti in alto e selezionare la voce RPC personalizzata.



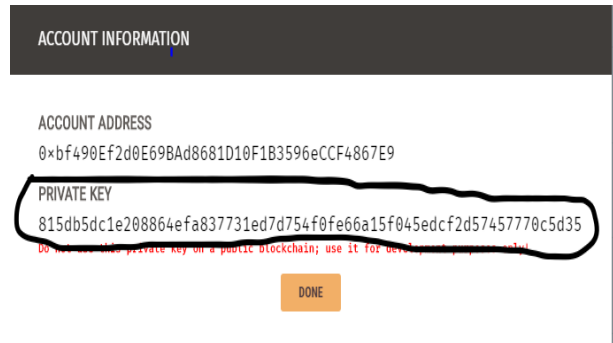
Compilare il form come da immagine e cliccare su Salva.

A screenshot of the Metamask 'Impostazioni' (Settings) form for adding a custom RPC network. The form is titled 'Impostazioni' and has a close button (X) in the top right corner. It contains the following fields: 'Nome Rete' (Network Name) with the value 'ganache', 'Nuovo URL RPC' (New RPC URL) with the value 'http://127.0.0.1:7545', 'ChainID (opzionale)' (Optional), 'Simbolo (opzionale)' (Optional), and 'URL del Block Explorer (opzionale)' (Optional). At the bottom, there are two buttons: 'Annulla' (Cancel) and 'Salva' (Save).

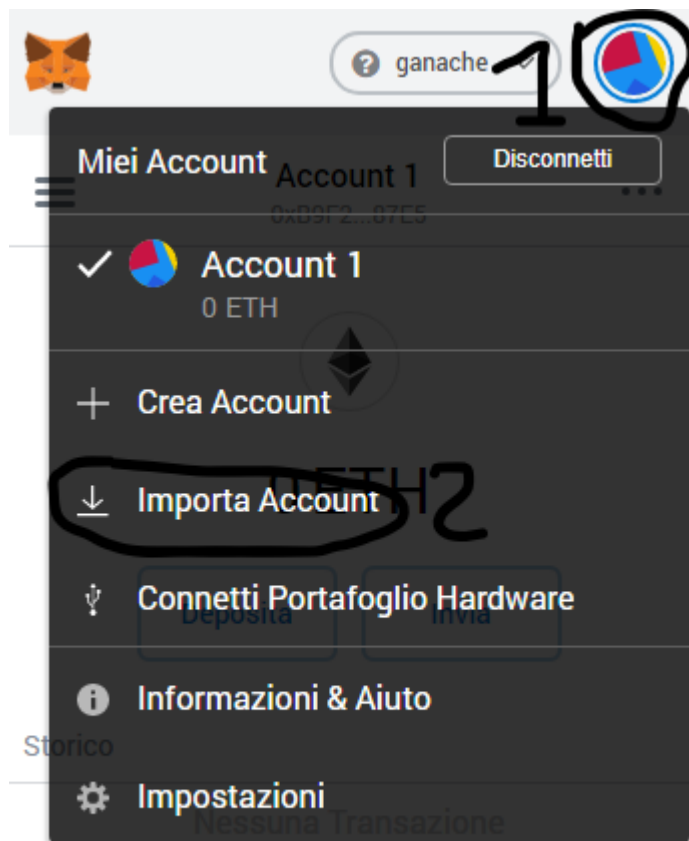
Aprire Ganache cliccare sul simbolo della chiave e copiare la chiave privata di un account



ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS	UPDATE AVAILABLE
<p>staircase proof excuse praise burden negative east cannot prefer select seminar</p> <p>HD PATH: a/44/90/90/90/account_index</p>						
ADDRESS: 0xbf490Ef2d0E69BAd8681D10F1B3596eCCF4867E9	BALANCE: 100.00 ETH	TX COUNT: 0	INDEX: 0	🔑		
ADDRESS: 0xC45F75f18d3156148A99982c3c1cc824f7b16A2C	BALANCE: 100.00 ETH	TX COUNT: 0	INDEX: 1	🔑		
ADDRESS: 0x12A3e6A68688ae85375f83e99775aFF110eFe7	BALANCE: 100.00 ETH	TX COUNT: 0	INDEX: 2	🔑		
ADDRESS: 0x645b4FE291b2881CcC455Aa71E3423485800c160	BALANCE: 100.00 ETH	TX COUNT: 0	INDEX: 3	🔑		
ADDRESS: 0x44015615E79F0302721C272D39c25321e8C42004	BALANCE: 100.00 ETH	TX COUNT: 0	INDEX: 4	🔑		
ADDRESS: 0x5e85DF18ca29748dAEda59d0b5c430A7b34460A9	BALANCE: 100.00 ETH	TX COUNT: 0	INDEX: 5	🔑		
ADDRESS: 0xFB2423358417976F48FaEfa9A422B65106430b6b	BALANCE: 100.00 ETH	TX COUNT: 0	INDEX: 6	🔑		



Andare su Metamask, in alto a destra cliccare sul cerchio colorato. Cliccare su importa account e inserire la chiave privata recuperata su ganache.



6.3 Home Page

Esplorando la pagina web possiamo utilizzare tutte le funzionalità. Registrati, Crea Evento, Check-in e Owner.

Registrati

< >

Crea Evento

< >

Check-in

< >

Owner

Nome

Nome Evento

Quantità in Ether

0

Tempo check-in in secondi

0

Crea Evento