UNIVERSITÀ
degli STUDI
di CATANIA

Dipartimento di Ingegneria Elettrica Elettronica Informatica

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Progetto di Internet of Things Based Smart Systems

Anno accademico 2023-2024

Studenti:

Gaia Natalj Contino 1000042354

Marianna Guzzardella 1000044123

1 Sommario

INTRODUZIONE

Il progetto ha lo scopo di memorizzare e leggere valori acquisiti da un nodo IoT senza fare uso di sistemi centralizzati di terze parti.
Utilizza una board di prototipazione IoT (Arduino Uno) per misurare una grandezza tramite un sensore di temperatura LM35 e memorizza i dati su blockchain.
La blockchain è un registro distribuito immutabile che consente la creazione di transazioni. Essa è costituita da una serie di blocchi concatenati, ognuno dei quali contiene un insieme di transazioni verificate. I blocchi sono collegati tramite funzioni crittografiche che rendono estremamente difficile la modifica di dati precedenti senza invalidare l'intera catena.
Alcune blockchain, come Ethereum, consentono la creazione e l'esecuzione di smart contract cioè programmi informatici autoeseguibili che consentono di automatizzare e regolare gli accordi digitali senza l'intervento di terze parti.
I software e i linguaggi utilizzati:

Software:

1. Arduino IDE: Per la creazione e il caricamento dello sketch sulla board.
2. WSL (Windows Subsystem for Linux): Distribuzione Linux su Windows, utilizzata per eseguire comandi e programmi Linux.
3. VS Code: IDE per la scrittura del codice.
4. Node.js: Runtime JavaScript utilizzato per eseguire codice lato server.
5. Yarn: Gestore di pacchetti JavaScript.
6. Usbipd: Programma per la condivisione di dispositivi USB tramite IP.
7. HardHat: Framework per lo sviluppo di smart contract su Ethereum.

Linguaggi utilizzati:

1. C++: Utilizzato nell'Arduino IDE per programmare la board Arduino Uno.
2. Solidity: Linguaggio di programmazione per smart contract su blockchain Ethereum.
3. JavaScript: Utilizzato per scrivere codice di backend specialmente per interagire con la blockchain Ethereum tramite Node.js.

1. Arduino Uno

Il seguente codice è un semplice sketch che consente la misurazione della temperatura in gradi Celsius grazie all'utilizzo del sensore di temperatura LM35.

```
#define PIN_LM35 A0

void setup() {
  Serial.begin(9600);
}

void loop() {
  int valore = analogRead(PIN_LM35);
  float temperatura = valore / 2.046;

  Serial.print("Temp.: ");
  Serial.print(temperatura);
  Serial.println("°C");

  delay(500);
}
```

1.1 Usbipd

Una difficoltà incontrata durante il progetto è stata la comunicazione tra Arduino Uno e WSL.
Poiché Arduino Uno non è dotato di moduli interni per la connessione Internet e non disponendo di ulteriori moduli esterni, non è stato possibile stabilire una comunicazione diretta tramite API tra le due parti. Di conseguenza, è stata adottata la connessione seriale per facilitare lo scambio di dati.
Inoltre, un'ulteriore problematica è stata la lettura della porta COM3 di Windows, alla quale è collegato Arduino, da WSL. Dopo una serie di ricerche e vari tentativi, è stata trovata una soluzione utilizzando Usbipd. Questo software agisce come un demone su un host USB, consentendo di gestire le richieste per la condivisione dei dispositivi USB su una rete tramite il protocollo USB/IP. In pratica, Usbipd è stato utilizzato per connettere la porta COM3 di Windows alla porta virtuale '/dev/ttyACM0' su WSL, permettendo così una comunicazione tra Arduino e WSL.

1.2 SerialPort

La lettura della porta avviene su JavaScript tramite la libreria SerialPort utilizzata per la comunicazione seriale con dispositivi collegati al computer. Questa libreria fornisce un'API per interagire con i dispositivi connessi e permette di enumerare e accedere alle porte seriali disponibili sul sistema.
La connessione avviene tramite la seguente funzione.

```
const leggiTemperatura = () => {
  const port = new SerialPort({ path: "/dev/ttyACM0", baudRate: 9600 });

  const parser = port.pipe(new ReadlineParser({ delimiter: "\n" }));

  return new Promise((resolve, reject) => {
    port.on("open", () => {
      console.log("Connessione seriale aperta");
    });

    parser.on("data", (data) => {
      console.log("Dato ricevuto da Arduino:", data);
      port.close();
      resolve(data);
    });

    port.on("error", (err) => {
      reject(err);
    });
  });
};
```

2. HardHat

Hardhat è un framework per lo sviluppo e il testing di smart contract su Ethereum. È progettato per semplificare il processo di sviluppo di applicazioni decentralizzate e contratti intelligenti sulla blockchain.

2.1 Solidity

Il contratto, chiamato "TemperaturaStorage", è scritto in Solidity, un linguaggio di programmazione ad oggetti progettato appositamente per la scrittura di contratti intelligenti su piattaforme blockchain.
Il contratto ha l'obiettivo di memorizzare valori di temperatura associati a timestamp su Ethereum.
Sono state implementate due funzioni:

1. **Store**: Funzione che consente di memorizzare un valore di temperatura nella mappa. Prende in input un timestamp e una temperatura, quindi aggiorna la mappa associando il timestamp alla temperatura corrispondente.
2. **Retrieve**:Funzione che consente di recuperare il valore di temperatura associato a un timestamp specifico dalla mappa. Prende in input un timestamp e restituisce la temperatura corrispondente associata a quella chiave.

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.8;

contract TemperaturaStorage {

    // Mappa che associa timestamp a valori di temperatura
    mapping(string => string) public temperatureData;

    // Funzione per memorizzare un valore di temperatura
    function store(string memory timestamp, string memory temperatura) public {
        temperatureData[timestamp] = temperatura;
    }

    // Funzione per recuperare il valore di temperatura associato a un timestamp
    function retrieve(string memory timestamp) public view returns (string memory) {
        return temperatureData[timestamp];
    }
}
```

2.2 Deploy

Il codice gestisce la distribuzione e l'utilizzo di un contratto Solidity chiamato "TemperaturaStorage" su una blockchain, insieme alla memorizzazione e al recupero dei dati di temperatura associati agli orari correnti.

```
async function main() {
  //Factory del contratto
  const contractFactory = await ethers.getContractFactory("TemperaturaStorage");
  console.log("Distribuzione del contratto...");
  // Deploy del contratto
  const contract = await contractFactory.deploy();
  await contract.deploymentTransaction();
  console.log(`Contratto distribuito all'indirizzo: ${contract.target}`);

  // Verifica del contratto su Etherscan
  if (network.config.chainId === 11155111 && process.env.ETHERSCAN_API_KEY) {
    await contract.deploymentTransaction().wait(6);
    await verify(contract.target, []);
  }
  try {
    const temperatura = await leggiTemperatura();
    console.log("Temperatura letta:", temperatura);
    const oraCorrente = date();
    console.log(oraCorrente);
    let t = await contract.retrieve(oraCorrente);
    console.log("Aggiornamento della temperatura...");
```

```
    // Creazione della transazione
    const transaction = await contract.store(oraCorrente, temperatura);
    await transaction.wait();
    console.log("Transazione avvenuta con successo!");
    const temperatura1 = await contract.retrieve(oraCorrente);
    console.log("Temperatura recuperata:", temperatura1);
  } catch (error) {
    console.error("Errore durante la lettura della temperatura:", error);
  }
```

2.3 Verifica

Verify è una funzione asincrona utilizzata per semplificare il processo di verifica dei contratti sulla blockchain utilizzando il plugin hardhat-etherscan, gestendo eventuali errori.
Utilizza il metodo run per eseguire il comando ‘verify:verify’ responsabile della verifica del contratto sull'esploratore di blocchi Etherscan.
Se si verifica un errore il codice controlla se il messaggio di errore contiene la stringa "already verified". Se sì, significa che il contratto è già stato verificato e viene stampato un messaggio informativo. Altrimenti, viene stampato l'errore stesso.

```
const verify = async (contractAddress, args) => {
  console.log("Verifica del contratto...");
  try {
    // Esegue la verifica del contratto utilizzando il plugin hardhat-etherscan
    await run("verify:verify", {
      address: contractAddress,
      constructorArguments: args,
    });
  } catch (e) {
    if (e.message.toLowerCase().includes("already verified")) {
      console.log("Contratto già verificato!");
    } else {
      console.log(e);
    }
  }
};
```

2.4 Task

Task personalizzato per Hardhat chiamato "block-number". Quando eseguito, stampa il numero del blocco corrente sulla console. Utilizza il modulo Hardhat per definire il comando e accedere al provider Ethereum per ottenere il numero del blocco corrente.

```
const { task } = require("hardhat/config");

//Definisce un nuovo comando personalizzato
task("block-number", "Prints the current block number").setAction(
  async (taskArgs, hre) => {
    const blockNumber = await hre.ethers.provider.getBlockNumber();
    console.log(`Current block number: ${blockNumber}`);
  }
);

module.exports = {};
```

3. Risultati ottenuti

Il contratto può essere distribuito tramite tre network differenti specificate all'interno del file hardhat.config.js.

```
defaultNetwork: "hardhat",
networks: {
  sepolia: {
    url: SEPOLIA_RPC_URL,
    accounts: [PRIVATE_KEY1],
    chainId: 11155111,
  },
  localhost: {
    url: "http://127.0.0.1:8545/",
    chainId: 31337,
  },
},
```

Tramite il comando di avvio è possibile specificare la rete su cui effettuare la transazione.
La rete hardhat è una rete locale fornita dal framework Hardhat stesso. Non è una vera e propria rete blockchain, ma una simulazione che gira localmente sulla macchina dello sviluppatore. È molto veloce e non richiede alcuna configurazione aggiuntiva per iniziare a sviluppare e testare i contratti.
La rete "localhost" fa riferimento all'ambiente locale di sviluppo sulla macchina.
È possibile configurare un nodo locale Ethereum utilizzando strumenti come Ganache o Geth e quindi collegare Hardhat a questo nodo per eseguire i test e lo sviluppo.
Entrambe queste reti sono utilizzate principalmente per lo sviluppo e il testing di contratti intelligenti prima della distribuzione su una rete blockchain live.

```
gaia@LAPTOP-AOPEO1H0:~/hardhat-LM35-Blockchain$ yarn hardhat run scripts/deploy.js --network hardhat
yarn run v1.22.15
$ /home/gaia/hardhat-LM35-Blockchain/node_modules/.bin/hardhat run scripts/deploy.js --network hardhat
Compiled 1 Solidity file successfully (evm target: london).
Distribuzione del contratto...
Contratto distribuito all'indirizzo: 0x5FbDB2315678afecb367f032d93F642f64180aa3
Connessione seriale aperta
Dato ricevuto da Arduino: Temp.: 25.90°C
Temperatura letta: Temp.: 25.90°C
22:27:25
Aggiornamento della temperatura...
Transazione avvenuta con successo!
Temperatura recuperata: Temp.: 25.90°C
Done in 7.97s.
gaia@LAPTOP-AOPEO1H0:~/hardhat-LM35-Blockchain$
```

3.1 Testing

I test servono a garantire che i contratti intelligenti si comportino come previsto durante lo sviluppo e la manutenzione del progetto su Ethereum.

```
describe("TemperaturaStorage", function () {
  let contractFactory, contract;
  const key = "k";
  beforeEach(async function () {
    contractFactory = await ethers.getContractFactory("TemperaturaStorage");
    contract = await contractFactory.deploy();
  });

  it("Should start with a value equal to the empty string", async function () {
    const currentValue = await contract.retrieve(key);
    const expectedValue = "";
    assert.equal(currentValue, expectedValue);
  });
  it("Should update when we call store", async function () {
    const expectedValue = "valoreTemperatura";
    const transactionResponse = await contract.store(key, expectedValue);
    await transactionResponse.wait(1);

    const currentValue = await contract.retrieve(key);
    assert.equal(currentValue.toString(), expectedValue);
  });
});
```

Gas Reporter di Hardhat è un plugin utilizzato per raccogliere e registrare dati sul gas utilizzato durante l'esecuzione dei test dei contratti intelligenti.

Per fare questo si utilizza CoinMarketCap tramite chiave API, necessaria per accedere ai servizi di CoinMarketCap e ottenere i tassi di cambio necessari per la conversione in euro del costo del gas.

gas-report.txt

```
·---------------------------------|---------------------------|-------------|-----------------------------·
|       Solc version: 0.8.8       ·  Optimizer enabled: false  ·  Runs: 200  ·  Block limit: 30000000 gas  |
··································|···························|·············|·····························
|  Methods                        ·               15 gwei/gas               ·       2094.78 eur/eth       |
·····················|···········|·············|·············|·············|··············|··············
|  Contract          ·  Method   ·  Min        ·  Max        ·  Avg        ·  # calls     ·  eur (avg)  |
·····················|···········|·············|·············|·············|··············|··············
|  TemperaturaStorage  ·  store  ·          -  ·          -  ·      46798  ·           2  ·       1.47  |
·····················|···········|·············|·············|·············|··············|··············
|  Deployments                    ·                                         ·  % of limit   ·             |
··································|·············|·············|·············|··············|··············
|  TemperaturaStorage             ·          -  ·          -  ·     422474  ·       1.4 %  ·      13.27  |
·---------------------------------|-------------|-------------|-------------|--------------|-------------·
```

3.2 Sepolia

Sepolia è una testnet, cioè una rete blockchain alternativa e separata dalla rete principale (mainnet) utilizzata per scopi di sviluppo, testing e sperimentazione. Le testnet sono progettate per consentire agli sviluppatori e agli utenti di testare le proprie applicazioni e contratti intelligenti senza dover utilizzare valuta criptografica reale e senza rischiare di perdere fondi.
Per la connessione tramite rete Sepolia si ha la necessita di utilizzare un wallet. In questo caso è stato usato MetaMask che è una delle estensioni più popolari per i browser web che consente agli utenti di accedere a blockchain Ethereum e a una serie di applicazioni decentralizzate direttamente tramite il proprio browser. MetaMask funge da portafoglio Ethereum virtuale, consentendo agli utenti di gestire le loro criptovalute ed inoltre gestisce le loro chiavi private in modo sicuro, consentendo loro di accedere ai loro fondi crittografici e di firmare transazioni direttamente dal browser.

```
gaia@LAPTOP-AOPEO1H0:~/hardhat-LM35-Blockchain$ yarn hardhat run scripts/deploy.js --network sepolia
yarn run v1.22.15
$ /home/gaia/hardhat-LM35-Blockchain/node_modules/.bin/hardhat run scripts/deploy.js --network sepolia
Distribuzione del contratto...
Contratto distribuito all'indirizzo: 0x1f38f9863cd1f642485b3F0d38D91BEA4f3EA913
Verifica del contratto...
Successfully submitted source code for contract
contracts/TemperatureStorage.sol:TemperaturaStorage at 0x1f38f9863cd1f642485b3F0d38D91BEA4f3EA913
for verification on the block explorer. Waiting for verification result...

Successfully verified contract TemperaturaStorage on the block explorer.
https://sepolia.etherscan.io/address/0x1f38f9863cd1f642485b3F0d38D91BEA4f3EA913#code

Connessione seriale aperta
Dato ricevuto da Arduino: Temp.: 26.39°C
Temperatura letta: Temp.: 26.39°C
22:56:47
Aggiornamento della temperatura...
Transazione avvenuta con successo!
Temperatura recuperata: Temp.: 26.39°C
Done in 106.83s.
```

Il valore di temperatura misurato e il relativo timestamp sono salvati nel campo dati del contratto e si possono visualizzare su Etherscan.
Etherscan è un servizio online che fornisce una vasta gamma di funzionalità e strumenti per esplorare, monitorare e analizzare l'attività sulla blockchain di Ethereum. Offre un'interfaccia utente intuitiva e potente che consente agli utenti di accedere a una serie di informazioni e dati relativi alla blockchain di Ethereum.

	Transaction Hash	Method	Block
	0x5ec2f0d6a47a52f62...	Store	5179684
	0x25a2d184413f26b8...	0x60806040	5179677

Txn Type: 2 (EIP-1559) Nonce: 86 Position In Block: 40

#	Name	Type	Data
0	timestamp	string	22:56:47
1	temperatura	string	Temp.: 26.39°C

Switch Back