Winning Software Solution

winningsoftwaresolution@gmail.com

Manuale Sviluppatore Shop Chain

Informazioni	
Redattori	WinningSoftwareSolution
Versione	1.0.0
Uso	esterno

Descrizione

Manuale sviluppatore.

Versione	Data	Redattore	Verificatore	Descrizione
1.0.0	07/05/2022	Federico Marchi	Elia Scandaletti	Stesura finale
0.1.0	04/04/2022	Federico Marchi	Matteo Galvagni	Stesura iniziale

Contents

1 Introduzione

Il presente documento ha la funzione di descrivere in dettaglio la procedura da seguire per l'installazione di tutte le componenti necessarie per il corretto funzionamento di Shop Chain. Si garantisce il corretto funzionamento del prodotto con le versioni indicate nel documento per i software utilizzati.
È necessario clonare la repository GitHub al seguente indirizzo: `https://github.com/iota97/WinningSoftwareSolution`.

2 Server

2.1 Requisiti

2.1.1 Node Js

Installare Node Js (v. 16.13.1) se non è già stato installato precedentemente. Una volta effettuata correttamente l'installazione, è necessario eseguire il seguente comando direttamente nella cartella `server` per l'installazione delle dipendenze:

```
$ npm install
```

2.1.2 Metamask

Per poter utilizzare la web app è necessario Metamask (v. 10.11.3).
Per avere il proprio wallet Metamask è possibile recarsi su metamask.io, installare l'estensione e scegliere se:

- **Creare un nuovo wallet**: è possibile creare un nuovo wallet selezionando "Create a Wallet" (Fig. 1 freccia blu). Durante la creazione del wallet ricordare di salvare la mnemonic phrase poiché necessaria nei passi successivi;
- **Importare un wallet già esistente**: è possibile anche importare un wallet già esistente selezionando l'opzione "Import wallet" (Fig. 1 freccia rossa). È sufficiente inserire la mnemonic phrase per poter utilizzare il proprio wallet su Metamask.

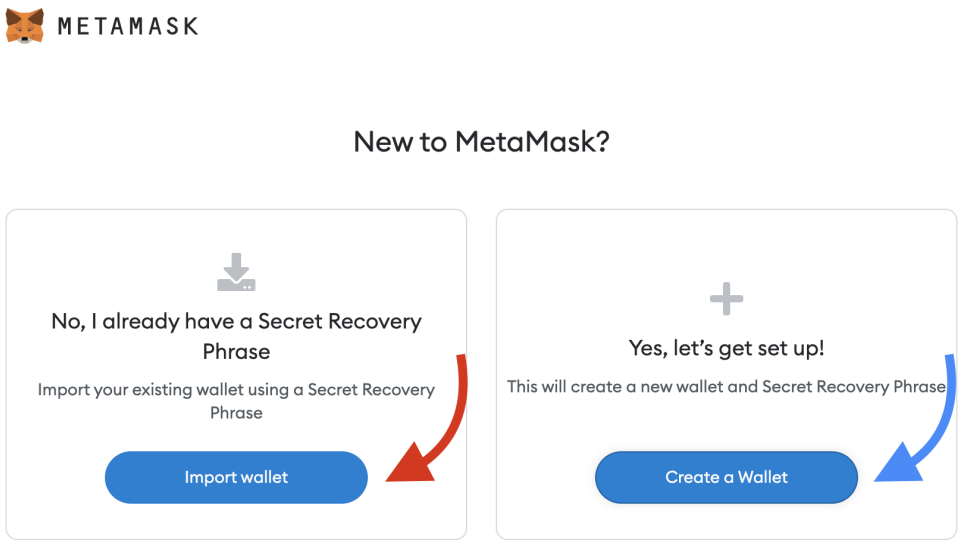

Figure 1: Creazione/Importazione wallet su Metamask.

Recupero mnemonic phrase

Nel caso in cui non si ricordi più la mnemonic phrase del proprio wallet è possibile recuperarla dal proprio account Metamask con la seguente procedura:

1. accedere al proprio wallet Metamask;
2. selezionare il proprio account in alto a destra (Fig. 2);

3. selezionare l'opzione "Settings";
4. selezionare l'opzione "Security & Privacy";
5. selezionare l'opzione "Reveal Secret Recovery Phrase";
6. inserire la propria password.

Una volta ottenuta la propria mnemonic phrase salvarla per i passi successivi.

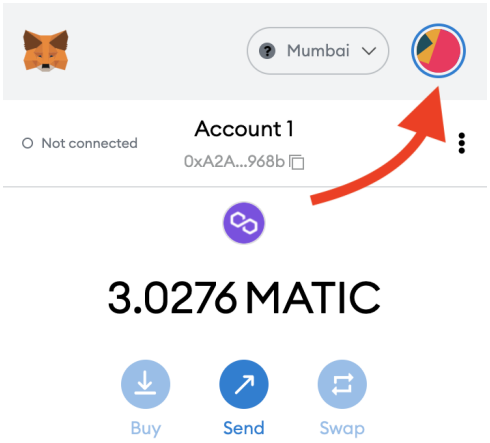

Figure 2: Selezione account Metamask.

Configurazione mainnet Polygon

Per l'inserimento e la configurazione della rete Polygon nel proprio wallet Metamask è necessario utilizzare i seguenti dati:

- Network Name: **Polygon**;
- New RPC Url: **https://rpc-mainnet.matic.network**;
- ChainID: **137**;
- Currency Simbol: **MATIC**;
- Block Explorer URL: **https://polygonscan.com/**.

Configurazione testnet Mumbai

Nel caso in cui si desidrasse operare sulla testnet Mumbai di Polygon, utilizzare i seguenti dati:

- Network Name: **Mumbai**;
- New RPC Url: **https://rpc-mumbai.maticvigil.com/**;
- ChainID: **80001**;
- Currency Simbol: **MATIC**;
- Block Explorer URL: **https://mumbai.polygonscan.com/**.

Nella testnet Mumbai è possibile eseguire qualsiasi operazione senza dover necessariamente spendere soldi. Infatti per rifornire il proprio wallet di MATIC di prova si può utilizzare il faucet all'indirizzo `https://faucet.polygon.technology/`.

Una volta inseriti i dati e aggiunta la rete, ricordare di selezionarla.

2.1.3 Provider

Per ottenere il provider per la mainnet Polygon è necessario registrarsi su moralis.io. Una volta effettuata la registrazione:

1. selezionare la sezione "SpeedyNodes" dal menù;
2. selezionare l'endpoint per Polygon Network;
3. selezionare il protocollo WS;
4. copiare l'url dell'endpoint per la mainnet (Fig. 3 freccia rossa).

Nell'url copiato è possibile visualizzare l' api key, si tratta infatti del codice evidenziato in grassetto nel seguente esempio:

wss://speedy-nodes-nyc.moralis.io/**a8d734a415dd368a3498db63**/polygon/mainnet/ws

È necessario salvare l'url per i passi successivi.

Testnet Mumbai

Nel caso in cui si volesse invece utilizzare una rete di test, è possibile utilizzare la rete Mumbai la quale è una testnet di Polygon. Per selezionare il provider corretto per Mumbai è sufficiente copiare l'url nel campo "Mumbai" (Fig. 3 freccia blu).

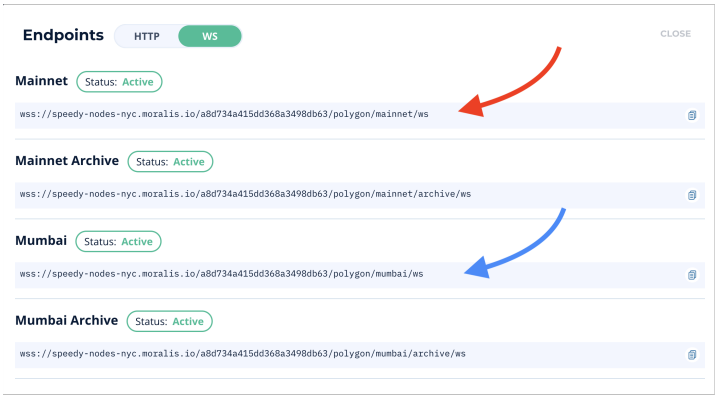

Figure 3: Provider su moralis.io.

2.1.4 MariaDB

Installare il database MariaDB (v. 10.7.3) se non è già stato installato precedentemente. Una volta eseguita l'installazione del database, collegarsi con il comando:

```
$ sudo mysql -u <user> -p <password>
```

e sostituire:

- `<user>`: con l'user con il quale si desidera creare il database;
- `<password>`: con la password dell'user selezionato.

Effettuato il collegamento eseguire la seguente query per la creazione del database:

```
CREATE DATABASE OnlineStore;
```

Successivamente creare le tabelle con le seguenti query:

```
USE OnlineStore;
DROP TABLE PaymentEntries;
DROP TABLE SettledPayments;
DROP TABLE LastBlockSynced;
CREATE TABLE PaymentEntries (id bigint, ecommerce varchar(255) not null,
price bigint not null, primary key(id));
CREATE TABLE SettledPayments (id bigint, item_id bigint not null,
buyer varchar(255) not null, status int not null, created bigint not null,
confirmed bigint, primary key(id));
CREATE TABLE LastBlockSynced (id int(1), value bigint not null, primary key(id));
INSERT INTO LastBlockSynced (id, value) VALUES (0, 0);
```

2.2 Configurazione

Una volta clonata la repository come spiegato nell'introduzione, è necessaria la creazione di un file '.env' dentro la cartella "Server". Il file deve essere strutturato come segue:

```
PORT=8080
DB_HOST="localhost"
DB_USER="username"
DB_PWD="password"
DB_NAME="OnlineStore"
API_KEY=xxxxxxxxxxxxxxxxxxxxxxxx
SERVER_URL="xxxxxxx.loca.ltx"
```

dove ciascun campo indica:

- **PORT**: la porta che si desidera utilizzare;
- **DB_HOST**: l'host selezionato;
- **DB_USER**: l'user associato al database;
- **DB_PWD**: password dell' user utilizzato;
- **DB_NAME**: il nome del database da utilizzare, in questo caso `OnlineStore`;
- **API_KEY**: l'api key, ottenibile come spiegato al punto 2.1.2.
- **SERVER_URL**: inserire il link ritornato da localtunnel come spiegato nel seguente paragrafo. Importante: l'URL va inserito senza il protocollo.

Sviluppo e testing con Localtunnel

Per lo sviluppo e il testing è stato utilizzato il tool localtunnel, il quale permette di rendere accessibile da remoto il server locale. Per utilizzare localtunnel è necessario:

- eseguire l'installazione globalmente con il comando:

```
$ npm install -g localtunnel
```

- avviare localtunnel con il comando:

```
$ lt --port <port>
```

con `<port>` la porta che si desidera utilizzare (deve corrispondere con la porta specificata nella variabile d'ambiente `PORT`);

– il link ritornato inserirlo nella variabile d'ambiente `SERVER_URL`. Importante ricordarsi di aggiornare la variabile ogni qualvolta si riavvia localtunnel.

2.3 Avvio server e test

Conclusa la configurazione è possibile far partire il server con il comando:

```
$ npm start
```

Mentre per i test è sufficiente la creazione del database `OnlineStoreTest` inserendo le seguenti query come spiegato al punto 2.1.3:

```
CREATE DATABASE OnlineStoreTest;
USE OnlineStoreTest;
DROP TABLE PaymentEntries;
DROP TABLE SettledPayments;
DROP TABLE LastBlockSynced;
CREATE TABLE PaymentEntries (id bigint, ecommerce varchar(255) not null,
price bigint not null, primary key(id));
CREATE TABLE SettledPayments (id bigint, item_id bigint not null,
buyer varchar(255) not null, status int not null, created bigint not null,
confirmed bigint, primary key(id));
CREATE TABLE LastBlockSynced (id int(1), value bigint not null, primary key(id));
INSERT INTO LastBlockSynced (id, value) VALUES (0, 0);
```

e successivamente eseguire il comando:

```
$ npm test
```

2.4 Test sulla Web App

Per eseguire i test sulla web app si utilizza JSCover in modalità proxy. I file Javascript della web app passano attraverso un server proxy che li trasforma e tiene traccia della loro copertura. Per eseguire correttamente il test seguire la seguente procedura:

- avviare il server di Shop Chain come spiegato al punto 2.3;
- avviare il server proxy con il comando:

  ```
  $ java -jar JSCover-all.jar -ws --proxy --port=3128
  --report-dir=jscoverage --local-storage
  ```

  è possibile cambiare porta nel caso la porta `3128` sia già occupata;
- configurare il proxy aggiungendo come host `localhost-proxy` e impostare come proxy `localhost-proxy:3128`;
- per eseguire i test è necessario connettersi a `http://localhost-proxy:8080`.

Salvataggio dati

JScover tiene traccia di ogni linea di codice eseguita e salva i dati nel localStorage di HTML5. Per salvare i dati nella cartella `jscoverage` è necessario:

- caricare la pagina `http://localhost-proxy:8080/jscoverage.html`;
- aprire la tab "Store";
- selezionare "Store Report".

Cancellazione dati

Essendo i dati dei test cumulativi, quando si desidera effettuare il test da zero è necessario cancellare i dati salvati. Per la cancellazione dei dati è sufficiente caricare la pagina `http://localhost-proxy:8080/jscoverage-clear-local-storage.html`.

Visualizzazione report

Per la visualizzazione del report è sufficiente selezionare la tab "Summary" nella seguente pagina: `http://localhost-proxy:8080/jscoverage-clear-local-storage.html`.

3 Script

Lo script in python permette all'e-commerce di creare una istanza di pagamento in blockchain e dunque permettere all'acquirente di acquistare attraverso il servizio Shop Chain.

3.1 Requisiti

3.1.1 Python e dipendenze

Al fine di utilizzare correttamente lo script è necessario:

- python3.8;
- pip3.8;
- installare le seguenti dipendenze con i comandi:

```
$ pip3.8 install web3
$ pip3.8 install python-dotenv
$ pip3.8 install pytest
```

3.2 Configurazione

È necessario creare un file chiamato '.env' all'interno della stessa cartella contenente lo script *sell.py*. Il file dovrà contenere:

- la mnemonic phrase del proprio wallet;
- il provider ottenuto al punto 2.1.2.

Di seguito un esempio del formato corretto da utilizzare:

```
MNEMONIC=mnemonic phrase del proprio wallet
PROVIDER=provider url
```

3.3 Creazione di un'istanza di pagamento

Una volta soddisfatti i requisiti e ultimata la configurazione è possibile procedere con l'esecuzione dello script python per la creazione di un'istanza di pagamento. Verranno illustrati due modi per la corretta esecuzione dello script *sell.py*.

3.3.1 Sell.py da terminale

È possibile eseguire lo script *sell.py* direttamente da terminale, attraverso il comando:

```
$ python3 sell.py [item price]
```

Sostituire `[item price]` con il prezzo con il quale si vuole creare l'istanza di pagamento. Una volta eseguito lo script, verrà ritornato e stampato l'id e il prezzo dell'istanza creata.
Esempio
Se vogliamo vendere un prodotto al prezzo di 2$, basterà inviare il comando:

```
$ python3 sell.py 2
```

l'output visualizzato sarà:

```
$ [Adding] Price: 2
$ [Added] Payment entry id: 3, price: 2
```

Dove :

- **paymement entry id: 3**: indica che l'id dell'oggetto messo in vendita è 3;
- **price: 2**: indica che il prezzo a cui è stato messo in vendita il prodotto è di 2$.

3.3.2 Script python

È possibile includere un proprio script in python nel backend dell' e-commerce per permettere direttamente la creazione delle istanze per i pagamenti.
La funzione che viene utilizzata è **sell_item** di *sell.py*. La funzione richiede in input un solo parametro, ovvero il prezzo in dollari che si desidera assegnare all'istanza di pagamento.
È dunque necessario includere nel proprio script la funzione come segue:

```
from sell import sell_item
```

3.4 Reindirizzamento alla landing page

Al fine di permettere all'acquirente, direttamente dal proprio e-commerce, di essere reindirizzato sulla landing page di Shop Chain per effettuare l'acquisto è necessario strutturare l'url di reindirizzamento come segue:

> `\land?id=<id_item>&r=<pagina_di_ritorno>`.

e dunque sostituire:

- `<id_item>` con l'id dell'istanza che si riferisce all'acquisto che l'acquirente desidera effettuare;
- `<pagina_di_ritorno>` con l'url della pagina alla quale si vuole reindirizzare l'acquirente al termine dell'acquisto.

Dopo aver confermato il pagamento l'acquirente verrà direttamente reindirizzato alla pagina di ritorno, l'url sarà strutturato come segue:

> `<pagina_di_ritorno>?transaction_id=<id_transazione>`.

dove `<id_transazione>` corrisponde all'id della transazione effettuata dall'acquirente per l'acquisto.

3.5 Test

Per eseguire il test sullo script è sufficiente eseguire il seguente comando:

```
$ pytest
```

4 Contract

4.1 Requisiti

4.1.1 Truffle, plugin e librerie

Per compilare lo smart contract e deployarlo è necessario installare Truffle e alcuni plugin e librerie all'interno della cartella `contract`. Eseguire i seguenti comandi per una corretta installazione:

```
$ npm install
$ npm install -g truffle
$ npm install @truffle/hdwallet-provider
```

per le librerie eseguire:

```
$ npm install @openzeppelin/contracts
$ npm install @chainlink/contracts
```

infine per la verifica e test installare:

```
$ npm install truffle-plugin-verify
$ npm install solidity-coverage
```

4.2 Configurazione

È necessaria la creazione di tre file nella cartella `contract` per il deployment del contratto:

- **mnemonic.secret**: deve contenere la mnemonic del proprio wallet;
- **providerlink.secret**: deve contenere l'url del provider ottenibile come spiegato al punto 2.1.2, può essere utilizzato anche l'url in http;
- **apikey.secret**: deve contenere l' api key necessaria per la verifica del contratto. Per ottenere l'api key registrarsi su PolygonScan, una volta eseguita la registrazione andare al seguente url per generare la propria api key: `https://polygonscan.com/myapikey`.

4.3 Compilazione, deployment e verifica

Compilazione

Per la compilazione del contract è sufficiente eseguire il seguente comando:

```
$ truffle compile
```

Deployment

Per il deployment del contratto eseguire il seguente comando:

```
$ truffle deploy --network <YOUR_NETWORK>
```

dove `<YOUR_NETWORK>` rappresenta il network che si desidera utilizzare, in questo caso viene utilizzato `polygon_mumbai`. Il network è definito nel file *truffle-config.js*.

In caso di errori di timeout cambiare l'url nel file *providerlink.secret* da http a wss e/o eseguire il comando aggiungendo:

```
$ truffle deploy --network <YOUR_NETWORK> --reset --compile-none
```

A questo punto è necessario registrare l'upkeeper di ChainLink per il proprio contratto al fine di poter utilizzare un timer per la scadenza degli ordini. Registrarsi dunque su `https://keepers.chain.link/` e seguire la procedura illustrata nella documentazione di ChainLink al seguente url: `https://docs.chain.link/docs/chainlink-keepers/register-upkeep/`.

Verifica

Per la verifica del contratto eseguire il comando:

```
$ truffle run verify ShopContract --network <YOUR_NETWORK>
```

questo comando ritorna un link a PolygonScan nel quale è possibile visualizzare il codice verificato e le ABI del contratto.

4.4 Test

Per testare senza copertura del codice eseguire il comando:

```
$ truffle test --network <YOUR_NETWORK> ./test/shopcontract.js
```

mentre per testare con copertura del codice eseguire il comando:

```
$ sudo truffle run coverage --file="./test/shopcontract.js"
  --solcoverjs ./.solcover.js
```