



Winning Software Solution

winningsoftwaresolution@gmail.com

ShopChain

SyncLab

Specifiche Architettureali

Informazioni

	Giovanni Cocco
<i>Redattori</i>	Elia Scandaletti
	Matteo Galvagni
<i>Revisori</i>	Federico Marchi
<i>Responsabili</i>	Giovanni Cocco
<i>Versione</i>	1.1.0
<i>Uso</i>	esterno

Descrizione

Architettura del progetto

Versione	Data	Persona	Attività	Descrizione
1.1.0	03/04/2022	E. Scandaletti, M. Galvagni	Redazione	Aggiunti dettagli dell'architettura
1.0.0	23/02/2022	Giovanni Cocco	Redazione	Creazione del documento

Contents

1	Introduzione	4
1.1	Scopo del documento	4
2	Riferimenti	4
2.1	Riferimenti normativi	4
2.2	Riferimenti informativi	4
3	Tecnologie/Linguaggi/Librerie	4
3.1	Solidity	4
3.2	Typescript	4
3.3	Express	4
3.4	MariaDB	4
3.5	Python	4
3.6	Web3	5
3.7	MetaMask	5
4	Architettura generale	5
4.1	Server	5
4.1.1	Persistenza	5
4.1.2	Server Web	5
4.2	Smart contract	6
4.3	Web app	6
4.4	Script di messa in vendita	6
5	Dettagli architettura	7
5.1	Server	7
5.1.1	Diagramma delle classi	7
5.1.2	Design pattern: Constructor injection	7
5.1.3	Schema DB	8
5.2	Smart contract	9
5.2.1	Diagramma delle classi	9
5.2.2	Lista dei metodi	9
5.2.3	Design pattern: Oracle	10
5.2.4	Design pattern: Access restriction	10
5.2.5	Design pattern: Guard check	11
5.2.6	Testing	11
5.3	Web app	12
5.3.1	Diagramma delle classi	12
5.3.2	Lista dei metodi	12
5.3.3	Design pattern: Nessuno	13
5.3.4	Testing	14
5.4	Script di messa in vendita	15
5.4.1	sell_item	15

6	Diagrammi di sequenza	16
6.1	Inizializzazione server web	16
6.2	Ascolto eventi del contratto	17
6.3	Nuovo oggetto in vendita	18
6.4	Nuova transazione	18
6.5	Cambio di stato di una transazione	19
6.6	Pagina transazioni in entrata	19

1 Introduzione

1.1 Scopo del documento

Il documento illustra le scelte architettureali e illustra l'architettura.

2 Riferimenti

2.1 Riferimenti normativi

- Capitolato d'appalto C2;
- Norme di Progetto;
- Verbale esterno 2022/03/01.

2.2 Riferimenti informativi

- Progettazione Software - Materiale didattico del corso IS;
- Slide diagrammi di sequenza - Materiale didattico del corso IS;
- Slide design pattern architetturali - Materiale didattico del corso IS;
- Slide diagrammi delle classi - Materiale didattico del corso IS;
- Slide principi SOLID - Materiale didattico del corso IS.

3 Tecnologie/Linguaggi/Librerie

3.1 Solidity

- **Versione:** 0.8.13
- **Documentazione:** <https://docs.soliditylang.org/en/v0.8.13/>

3.2 Typescript

- **Versione:** 4.6.3
- **Documentazione:** <https://www.typescriptlang.org/docs/>

3.3 Express

- **Versione:** 4.17.2
- **Documentazione:** <https://devdocs.io/express/>

3.4 MariaDB

- **Versione:** 10.7.3
- **Documentazione:** <https://mariadb.com/kb/en/documentation/>

3.5 Python

- **Versione:** 3.8
- **Documentazione:** <https://docs.python.org/3.8/>

3.6 Web3

- **Versione:** 1.7.1
- **Documentazione:** <https://web3js.readthedocs.io/en/v1.7.1/>

3.7 MetaMask

- **Versione:** 10.11.3
- **Documentazione:** <https://docs.metamask.io/guide/>

4 Architettura generale

Il progetto si compone di 4 macro parti:

- Server
- Smart contract
- Web app
- Script di messa in vendita

4.1 Server

Realizzato in typescript con express come modulo http e MariaDB come database SQL. Si divide in 2 parti principali: la persistenza e il server web.

4.1.1 Persistenza

Si occupa di gestire i dati delle transazioni.

Si collega allo smart contract attraverso un websocket fornito da moralis.io.

Rimane in ascolto degli eventi dello smart contract e aggiorna il database SQL di conseguenza.

Tiene sempre traccia dell'ultimo blocco da cui ha ricevuto un evento e all'avvio recupera tutti gli eventi arretrati partendo da quest'ultimo blocco.

I dati nel database SQL vengono forniti al frontend.

Notare come è molto oneroso effettuare query sui dati in block chain in quanto non sono disponibili strutture dati adeguate.

Questa soluzione ci permette una maggiore flessibilità e apertura a modifiche future quali aggiungere query specifiche.

Inoltre il contratto una volta pubblicato non può essere modificabile al fine di garantire la trasparenza ed è quindi cruciale che il codice di quest'ultimo sia semplice ed affidabile.

4.1.2 Server Web

Riceve le richieste HTTP dalla rete e risponde con le pagine della Web app.

4.2 Smart contract

Realizzato in solidity e pubblicato sulla rete Polygon tiene traccia delle transazioni; gestisce la logica e la sicurezza di esse.

Per gestire il timer che fa scadere le transazioni si usa il servizio Upkeep di ChainLink. Uno smart contract non esegue operazioni se non viene chiamato, per realizzare un timer si realizzano delle funzioni che eseguono le operazioni necessarie se è passato abbastanza tempo. Un upkeep chiama automaticamente queste funzioni dall'esterno a intervalli di tempo prefissato.

4.3 Web app

Fornita all'utente tramite il server web fornisce la logica lato client. Tramite MetaMask l'utente interagisce direttamente col contratto.

In caso di utenti mobile reindirizza tramite deep link per aprire la pagine sull'app di Metamask.

I deep link vengono usati anche per il QR code di ricezione del pacco. Possono essere scansionati sia da dentro l'app di MetaMask che dalla fotocamera del cellulare.

4.4 Script di messa in vendita

Usato dall'e-commerce per inserire prodotti in vendita in blockchain al fine di verificare che il prezzo sia corretto al momento della vendita. Realizzato in python per flessibilità, si connette alla block chain tramite moralis.io.

5 Dettagli architettura

5.1 Server

5.1.1 Diagramma delle classi

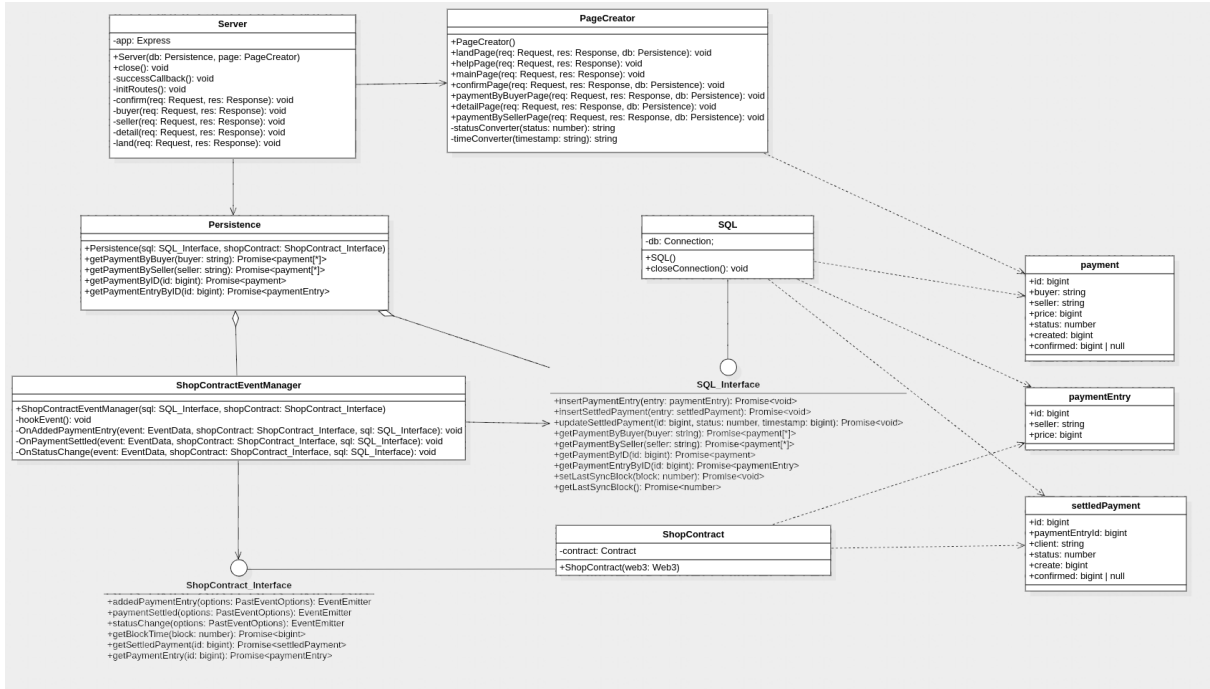


Figure 1: Diagramma delle classi del server

Commenti

La classe ShopContract andrà a interfacciarsi con il contratto in blockchain.
La classe PageCreator andrà a interfacciarsi con la WebApp.

5.1.2 Design pattern: Constructor injection

Descrizione

Le dipendenze sono tracciate e passate agli oggetti tramite il costruttore.

Motivazioni

Facilita il tracciamento delle dipendenze e agevola il mocking in fase di test.

5.1.3 Schema DB

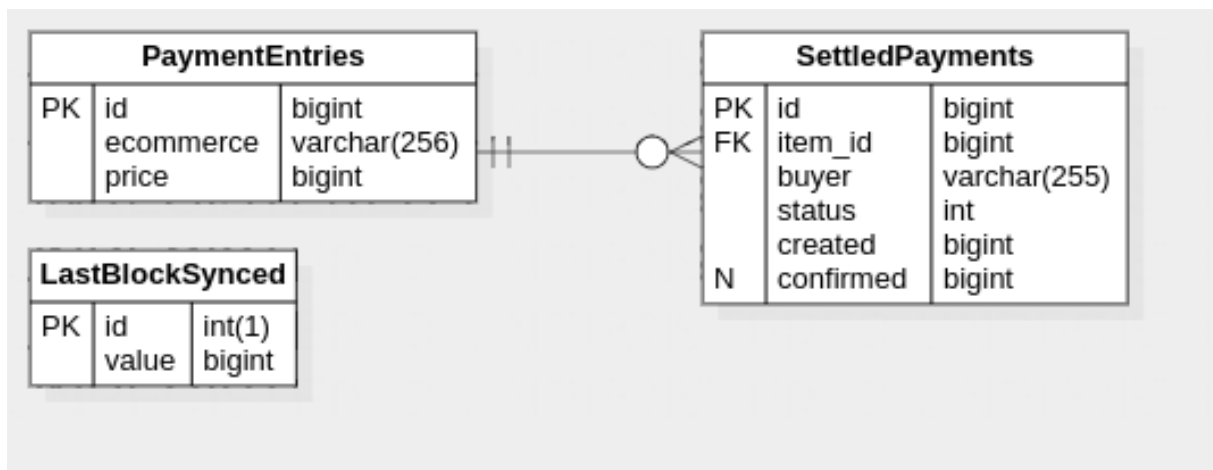


Figure 2: Schema del DB relazionale

Commenti

Schema delle tabelle del database.

LastSyncedBlock contiene una sola riga con *id* 0 con il valore dell'ultimo blocco sincronizzato.

5.2 Smart contract

5.2.1 Diagramma delle classi

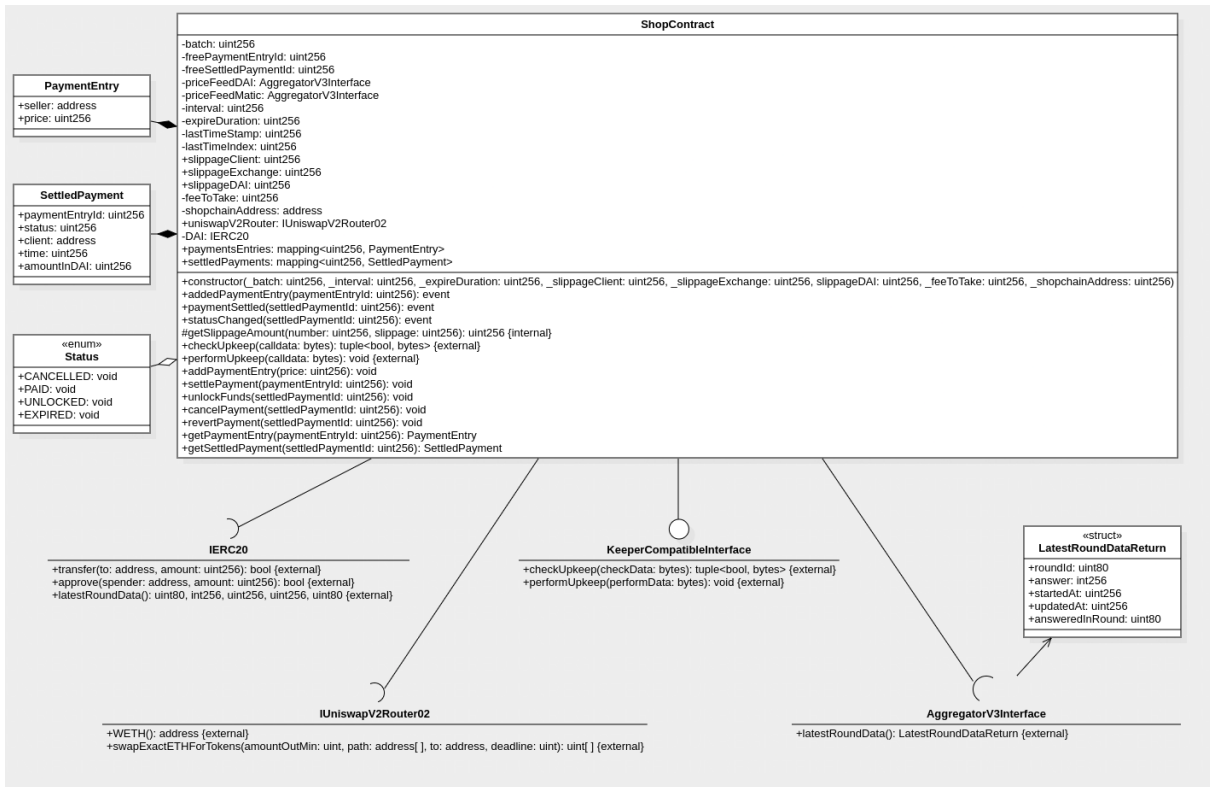


Figure 3: Diagramma delle classi del contratto

Commenti

In Solidity esistono due particolari visibilità di funzione che differiscono da quelle normali: *internal* ed *external*. Una funzione *internal* può essere chiamata solo dal contratto in cui è dichiarata e da contratti derivanti. È l'equivalente di *protected*. Una funzione *external* può essere chiamata solo dall'esterno e non all'interno del contratto stesso.

5.2.2 Lista dei metodi

- **constructor(_interval: uint256, _expireDuration: uint256, _slippageClient: uint256, _slippageExchange: uint256, _slippageDAI: uint256, _feeToTake: uint256, _shopchainAddress: address): void**

Imposta l'intervallo di controllo per aggiornamenti e l'intervallo di scadenza. Imposta gli slippage, la percentuale di fee da trattenere e l'address a cui inviarla. Imposta router e factory dell'exchange su cui convertire Matic in DAI.

- **checkUpkeep(calldata: bytes): (upkeepNeeded: bool, memory: bytes)**
Se chiamata, restituisce tramite *upkeepNeeded* se è passato l'intervallo di tempo minimo dall'ultimo upkeep e dunque se è necessario eseguirne un altro.
- **performUpkeep(calldata: bytes): void** Controlla se l'intervallo di tempo minimo per ogni upkeep è passato, se è così scorre dall'ultima transazione vista alla più recente transazione che abbia superato la durata massima senza sblocco

dei fondi e restituisce il denaro all'indirizzo del cliente.

- **addPaymentEntry(price: uint256): void** Aggiunge un'entry di pagamento alla relativa mappa.
- **settlePayment(paymentEntryId: uint256): void** Completa il pagamento dato l'id di un pagamento aperto, eseguendo i necessari controlli e aggiungendo un nuovo pagamento completato alla relativa mappa.
- **unlockFunds(settledPaymentId: uint256): void** Dato l'id di un pagamento completato, se chiamata dal cliente originale e dopo necessari controlli sblocca i fondi che vengono inviati all'indirizzo del venditore.
- **cancelPayment(settledPaymentId: uint256): void** Se il venditore avesse necessità di annullare una transazione dopo il pagamento da parte del cliente, questa funzione si occuperà di restituire il denaro al cliente.
- **revertPayment(settledPaymentId: uint256): void** Se il servizio di UpKeep dovesse smettere di funzionare per qualsiasi motivo, questa funzione consente al cliente di ottenere un rimborso manualmente qualora il pacco non sia arrivato dopo l'intervallo prefissato.
- **getPaymentEntry(paymentEntryId: uint256): PaymentEntry** Ritorna l'oggetto rappresentante un'entry di pagamento aperta da un venditore.
- **getSettledPayment(settledPaymentId: uint256): SettledPayment** Ritorna l'oggetto rappresentante un pagamento completato da un cliente relativo ad un'entry di pagamento di un venditore.

5.2.3 Design pattern: Oracle

Descrizione

Le informazioni sul reale valore di una valuta sono fornite da un oracolo.

Motivazioni

Per evitare manipolazioni temporanee del valore della valuta stabile utilizzata si utilizza il valore fornito da ChainLink. ChainLink restituisce la media del valore di una valuta tra più fornitori, evitando manipolazioni di un singolo fornitore.

5.2.4 Design pattern: Access restriction

Descrizione

La chiamata di alcune funzioni è ristretta ad alcune tipologie di utenti.

Motivazioni

È necessario mantenere una distinzione tra cliente e venditore all'interno del contratto. Il venditore non potrà chiamare funzioni riservate al cliente (per esempio, sbloccandosi i fondi) e viceversa.

5.2.5 Design pattern: Guard check

Descrizione

L'input fornito alle funzioni è validato prima di essere processato, annullando la transazione se non valido.

Motivazioni

Alcune funzioni richiedono il passaggio in input di un ID. Deve essere controllato che tale ID esista prima di continuare le operazioni.

5.2.6 Testing

Per testare il contratto si utilizza la suite Truffle (usata anche per compilazione e deployment). Al fine di includere nei test di unità la copertura del codice, viene utilizzato il plugin solidity-coverage alla versione 0.6.3 (documentazione al link <https://github.com/sc-forks/solidity-coverage/tree/0.6.x-final#solidity-coverage>).

Il plugin clona la blockchain di test Mumbai di Polygon e la esegue in locale, per poi effettuare il deployment del contratto ed eseguire i test.

Di comune accordo col Proponente i test che richiedono il mocking completo di altri contratti non sono stati eseguiti visto l'alto costo di implementazione e la bassa utilità.

5.3 Web app

5.3.1 Diagramma delle classi

Non sono presenti classi. Questo perché introdurre delle classi avrebbe aumentato la complessità del codice, a fronte di vantaggi irrilevanti. Questo perché la web app è molto semplice ed è una mera interfaccia asincrona per la comunicazione col contratto in blockchain.

5.3.2 Lista dei metodi

Nessuno dei seguenti metodi prevede alcuna verifica sull'identità dell'utente o sulla validità delle richieste fatte. Questo perché la web app serve solo a fare da interfaccia per la blockchain che è pubblica per definizione. Tutta la parte di sicurezza è implementata nel contratto. Inoltre, essendo lo script eseguito lato client, sarebbe facile per un attaccante bypassare qualsiasi controllo.

- **mobileCheck(): boolean** Controlla se sta venendo usato un dispositivo mobile.
- **toggleMenu(): void** Mostra o nasconde il menu nella versione mobile dell'app.
- **setGetParameter(paramName: string, paramValue: string): void** Imposta un dato parametro GET nella barra dell'URL del browser.
- **findGetParameter(parameterName: string): string** Ritorna un dato parametro GET dall'URL corrente.
- **handleNewChain(id: string): void** Funzione utilizzata per gestire il cambio di blockchain a cui è collegato MetaMask.
- **handleNewAccounts(newAccounts: string[]): void** Funzione utilizzata per gestire il cambio di wallet utilizzati in MetaMask.
- **async isMetamaskConnected(): boolean** Controlla se MetaMask è connesso a un wallet.
- **showSellerButton(): void** Nella parte di web app relativa al venditore, mostra il bottone di annullamento dell'ordine insieme al QR code.
- **showBuyerButton(): void** Nella parte di web app relativa all'acquirente, mostra il bottone di sblocco manuale del pagamento.
- **showContents(): void** Nasconde i pop-up e mostra il contenuto della pagina.
- **updateMenuLink(): void** Aggiorna i link nel menù in modo che puntino alle pagine relative al wallet attualmente collegato.
- **async onMetamaskConnected(): void** Procedura asincrona che viene eseguita alla connessione di MetaMask a un wallet.

- **onClickInstall(): void** Procedura per l'installazione di MetaMask.
- **async onClickConnect(): void** Procedura per la connessione a MetaMask.
- **async onClickOpenMetaMask(): void** Procedura per aprire la web app nell'applicazione mobile di MetaMask.
- **isMetaMaskInstalled(): boolean** Controlla se MetaMask è installato.
- **askForInstall(): void** Mostra un pop-up per chiedere all'utente di installare MetaMask.
- **askToConnect(): void** Mostra un pop-up per chiedere all'utente di collegare il proprio wallet a MetaMask.
- **async initialize(): void** Funzione di inizializzazione che verifica se MetaMask è installato e collegato a un wallet e, in caso negativo, mostra i rispettivi pop-up per guidare l'utente nella risoluzione del problema.
- **closePop(e: Event): void** Procedura per chiudere il pop-up dato.
- **checkChainID(): boolean** Controlla che MetaMask sia collegata alla stessa blockchain del contratto.
- **showStatus(id: int): void** Mostra il pop-up corrispondente all'esito della transazione in corso o appena effettuata.
- **onClickSettlePayment(): void** Procedura per invocare il metodo del contratto `settlePayment`.
- **onClickCancelPayment(): void** Procedura per invocare il metodo del contratto `cancelPayment`.
- **onClickUnlockFunds(): void** Procedura per invocare il metodo del contratto `unlockFunds`.

5.3.3 Design pattern: Nessuno

Descrizione

Non è stato adottato alcun design pattern.

Motivazioni

Adottare pattern architetturali avrebbe richiesto un costo ben superiore ai benefici. Questo perché la web app è molto semplice ed una mera interfaccia asincrona per la comunicazione col contratto in blockchain.

5.3.4 Testing

Per eseguire i test sulla web app si è scelto di utilizzare JSCover in modalità proxy. In altre parole, i file js dell'applicazione passeranno attraverso un server proxy che li trasformerà e terrà traccia della loro copertura.

Per eseguire i test sarà necessario:

1. avviare il server della web app;
2. avviare il server di JSCover;
3. configurare il proxy;
4. eseguire e salvare i test.

Avvio del server ShopChain

Per avviare il server proxy eseguire il seguente comando:

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

È possibile cambiare porta nel caso la 3128 sia già occupata.

Configurazione del proxy

Il server da raggiungere è un server locale ma spesso le impostazioni del browser o del sistema operativo impediscono l'uso del proxy quando ci si collega a `localhost` o a `127.0.0.1`.

Per questo motivo è necessario aggiungere un nuovo host chiamato `localhost-proxy` alla lista degli host noti (`/etc/hosts` nei sistemi Unix) e impostare come proxy `localhost-proxy:3128`.

Esecuzione dei test

Per eseguire i test sarà necessario connettersi a `http://localhost-proxy:8080`.

Salvataggio dei dati JSCover terrà traccia di ogni linea di codice eseguita e salverà i dati nel `localStorage` di HTML5.

Per salvare i dati nella cartella `jscoverage` sarà necessario:

1. caricare la pagina `http://localhost-proxy:8080/jscoverage.html`;
2. aprire la tab "Store";
3. cliccare su "Store Report".

Cancellazione dei dati I dati dei test sono cumulativi, perciò quando bisogna effettuare i test da zero è necessario cancellare i dati salvati.

Per cancellare i dati è necessario caricare la pagina `http://localhost-proxy:8080/jscoverage-clear-local-storage.html`.

Vedere i report Per visualizzare il report è sufficiente:

1. caricare la pagina `http://localhost-proxy:8080/jscoverage.html`;
2. aprire la tab "Summary".

5.4 Script di messa in vendita

5.4.1 `sell_item`

Parametri

price: float - il prezzo in dollari della entry di pagamento.

Valore Restituito

entry id: int - l'id dell'entry inserita in blockchain (-1 in caso di errore).

Comportamento

Il metodo inserisce una nuova entry di pagamento in blockchain con il prezzo specificato. Sia in caso di successo che di errore tutte le informazioni vengono salvate nel file *sell.log*.

6 Diagrammi di sequenza

6.1 Inizializzazione server web

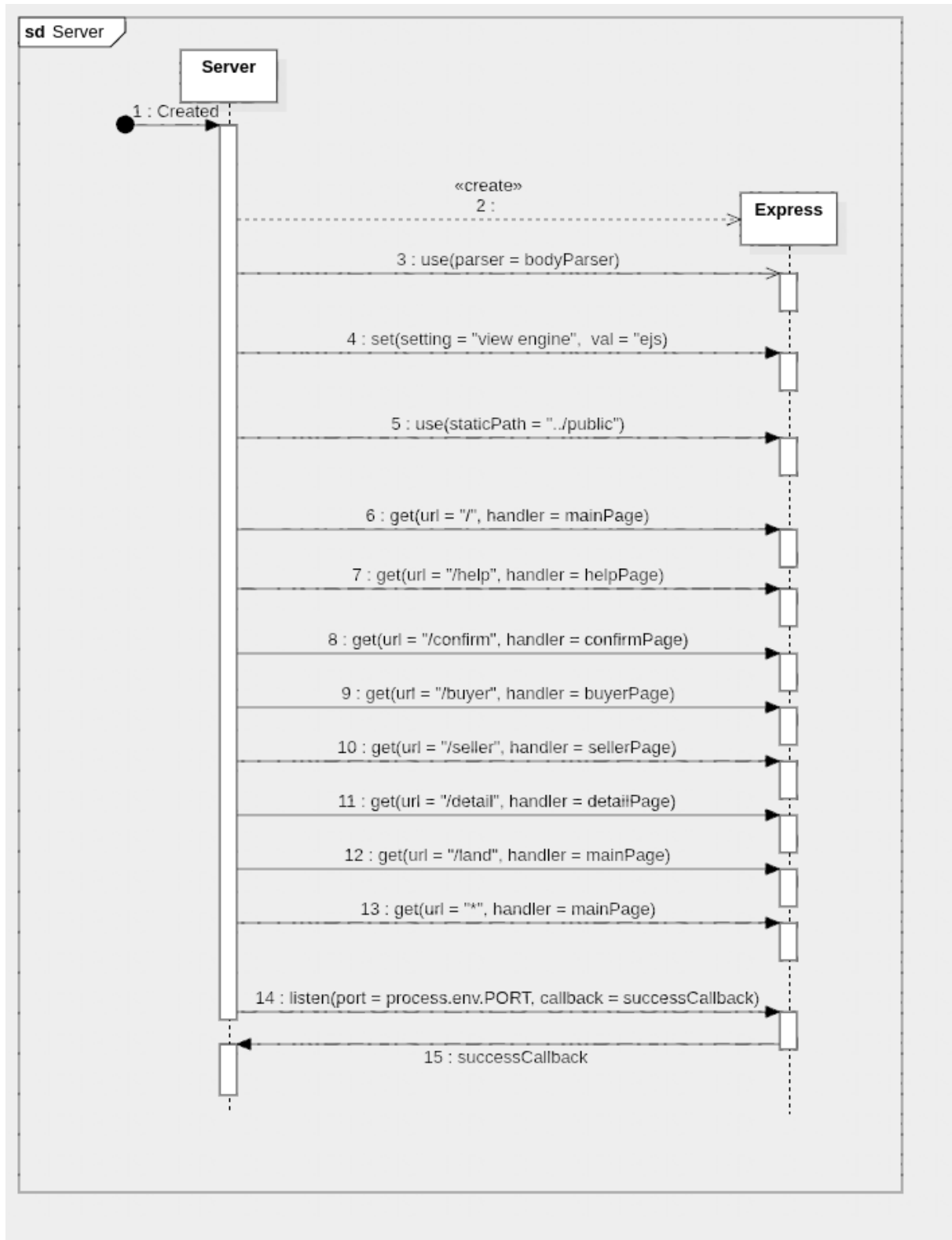


Figure 4: Diagramma di sequenza dell'inizializzazione del server

Commenti

Mostra l'inizializzazione delle routes per express.

6.2 Ascolto eventi del contratto

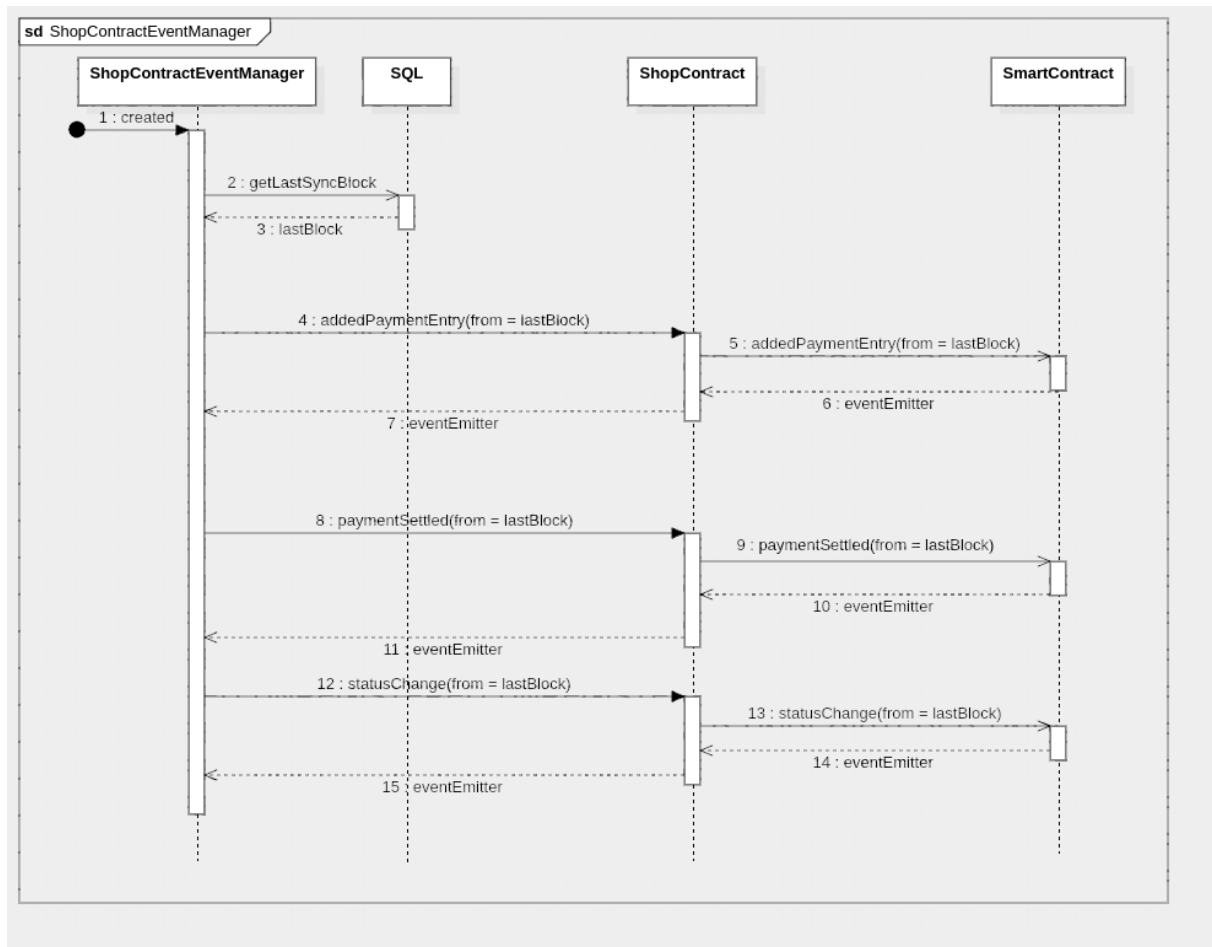


Figure 5: Diagramma di sequenza dell'ascolto degli eventi

Commenti

Mostra la sottoscrizione degli eventi del contratto.

6.3 Nuovo oggetto in vendita

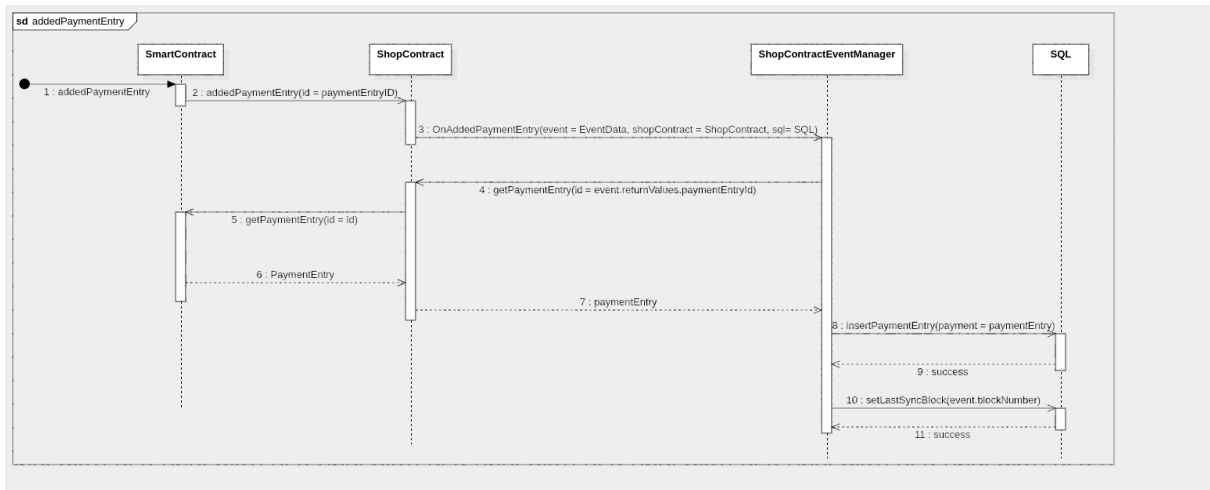


Figure 6: Diagramma di sequenza di un nuovo oggetto in vendita

Commenti

Mostra cosa succede quando viene inserita una nuova entry di pagamento da parte di un e-commerce.

6.4 Nuova transazione

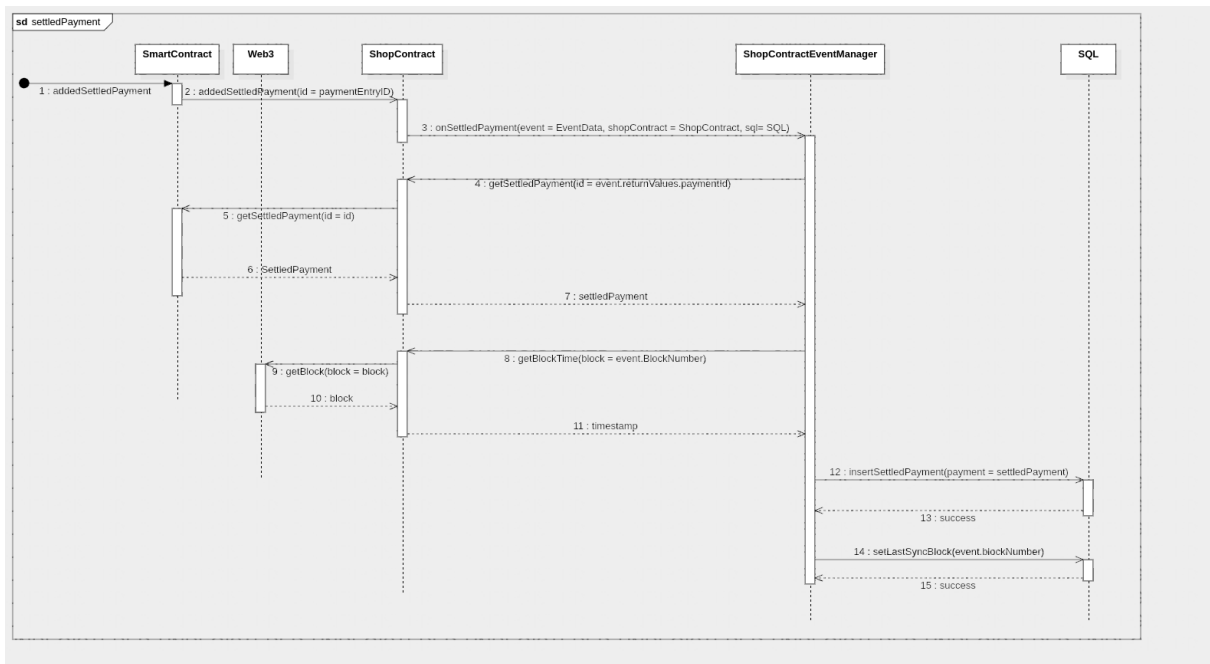


Figure 7: Diagramma di sequenza di una nuova transazione

Commenti

Mostra cosa succede quando viene creata una nuova transazione.

6.5 Cambio di stato di una transazione

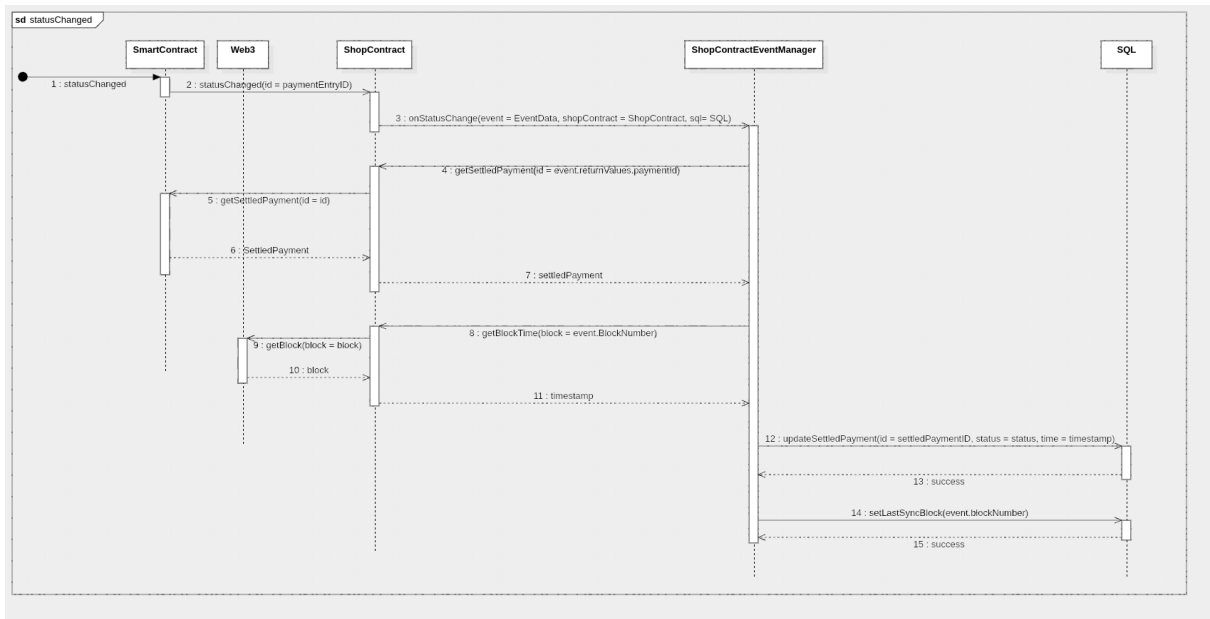


Figure 8: Diagramma di sequenza del cambio di stato di una transazione

Commenti

Mostra cosa succede quando una transazione cambia di stato.

6.6 Pagina transazioni in entrata

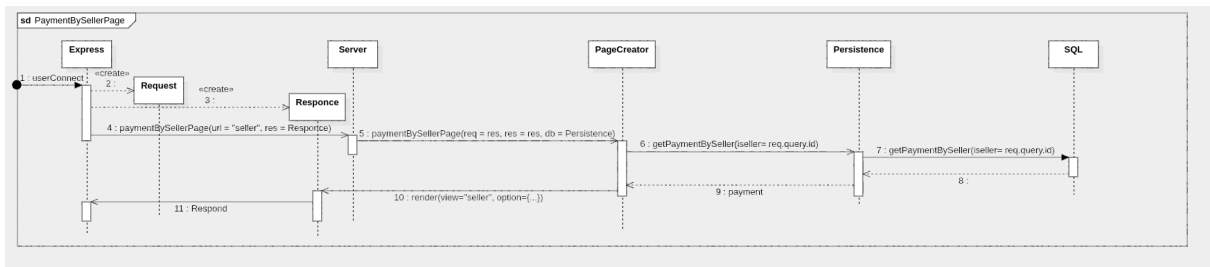


Figure 9: Diagramma di sequenza della richiesta della pagina delle transazioni in entrata

Commenti

Mostra cosa succede quando un utente richiede la pagina delle transazione in entrata. Le altre pagine utilizzano lo stesso modello e dunque si preferisce evitare diagrammi di sequenza ridondanti.