Winning Software Solution
winningsoftwaresolution@gmail.com

ShopChain
SyncLab

Specifiche Architetturali

Informazioni	
	Giovanni Cocco
Redattori	Elia Scandaletti
	Matteo Galvagni
Revisori	Federico Marchi
Responsabili	Giovanni Cocco
Versione	1.0.0
Uso	esterno

Descrizione
Architettura del progetto

Versione	Data	Persona	Attivtà	Descrizione
1.0.0	23/2/2022	Giovanni Cocco	Redazione	Creazione del documento

Contents

1 Introduzone

1.1 Scopo del documento

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

2 Riferimenti

2.1 Riferimenti normativi

- Capitolato d'appalto C2;
- Norme di Progetto;
- Verbale esterno 2021/03/01.

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 publicato 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 upkeeper 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'ecommerce 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

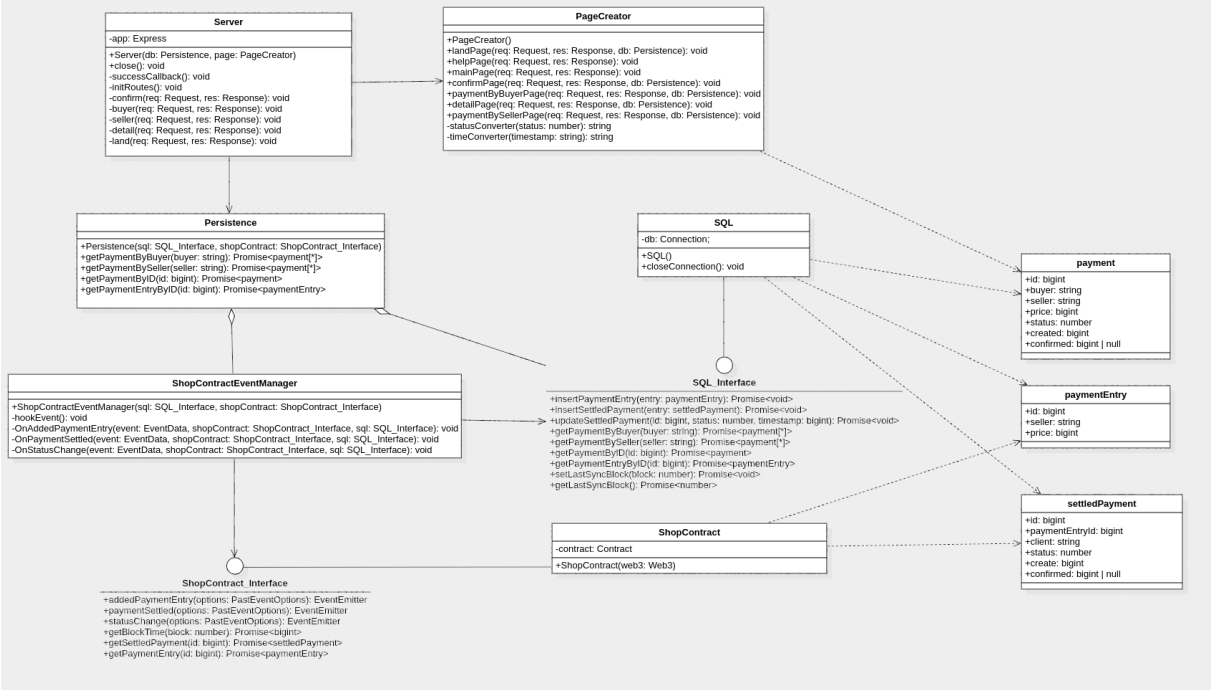

Figure 1: Diagramma delle classi del server

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 construttore.

Motivazioni

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

5.1.3 Schema DB

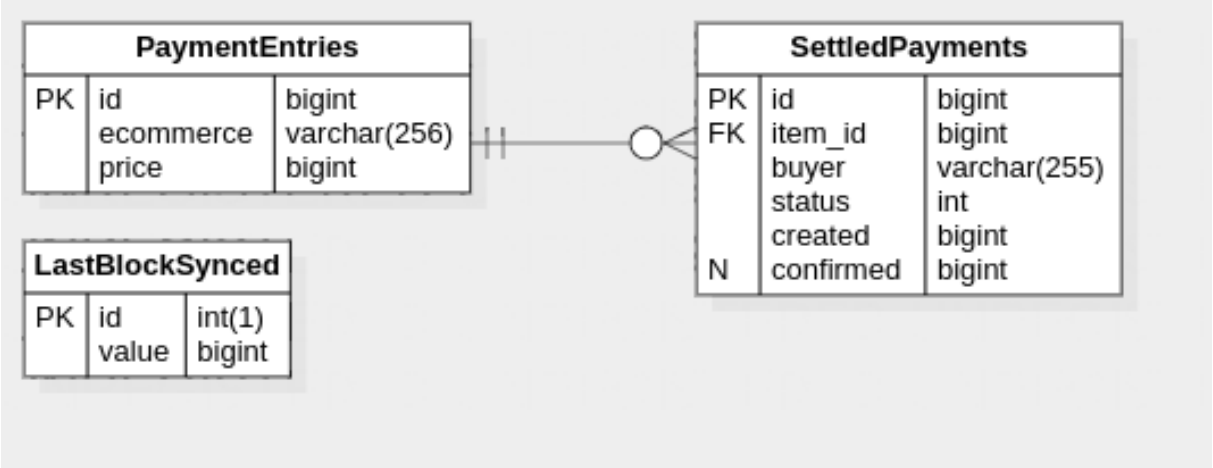

Figure 2: Schema del DB relazionale

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. 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 performarne 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.

5.3 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 irrisori. Questo perché la web app è molto semplice ed una mera interfaccia asincrona per la comunicazione col contratto in blockchain.

5.3.2 Lista dei metodi

TODO

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.

5.4 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 si successo che di errore tutte le informazione vengono salvate nel file *sell.log*.

6 Diagrammi di sequenza

6.1 Inizializzazione server web

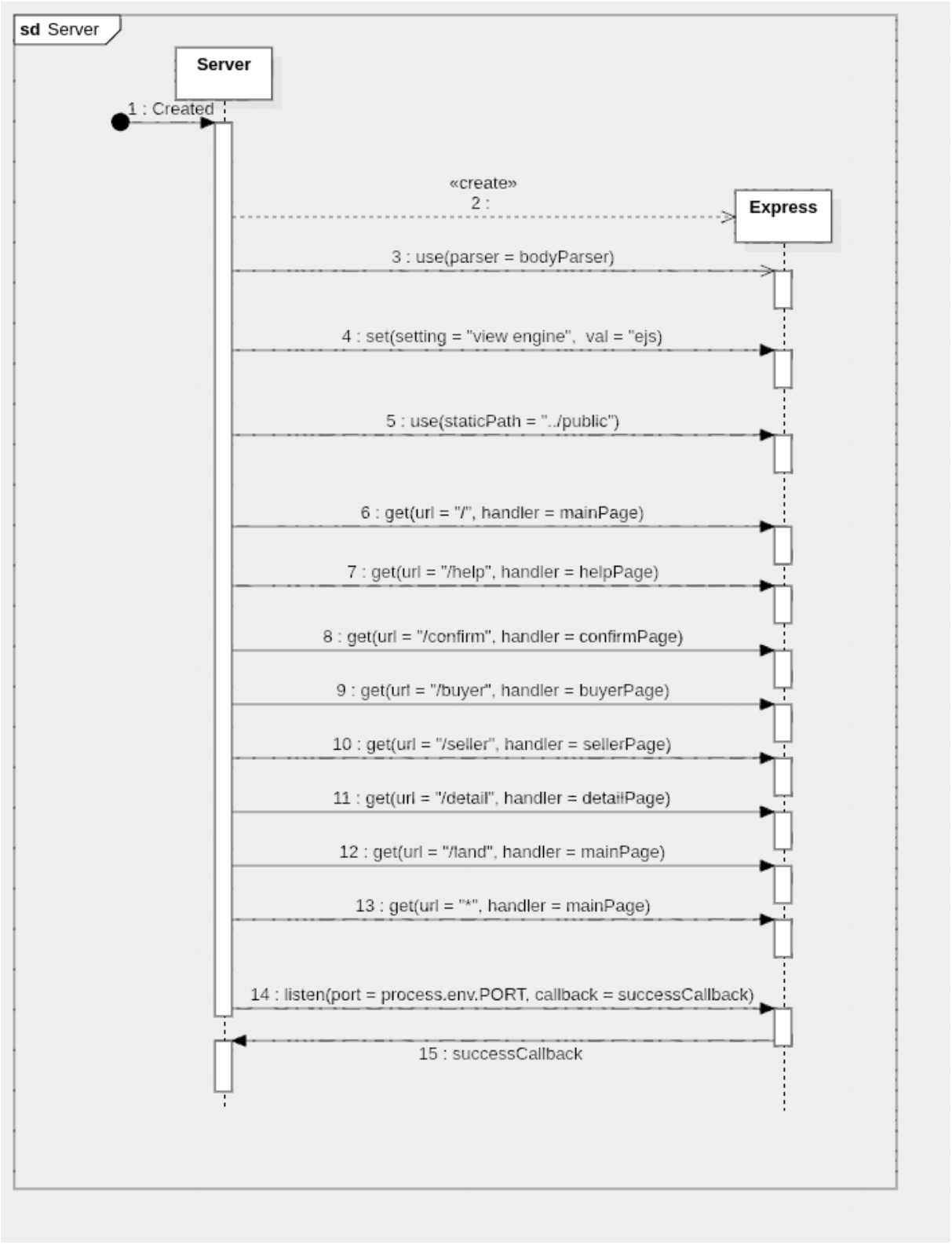

Figure 4: Diagramma di sequenza dell'inizializzazione del server

Commenti

Mostra l'inizializzazione delle routes per express.

6.2 Ascolto eventi del contratto

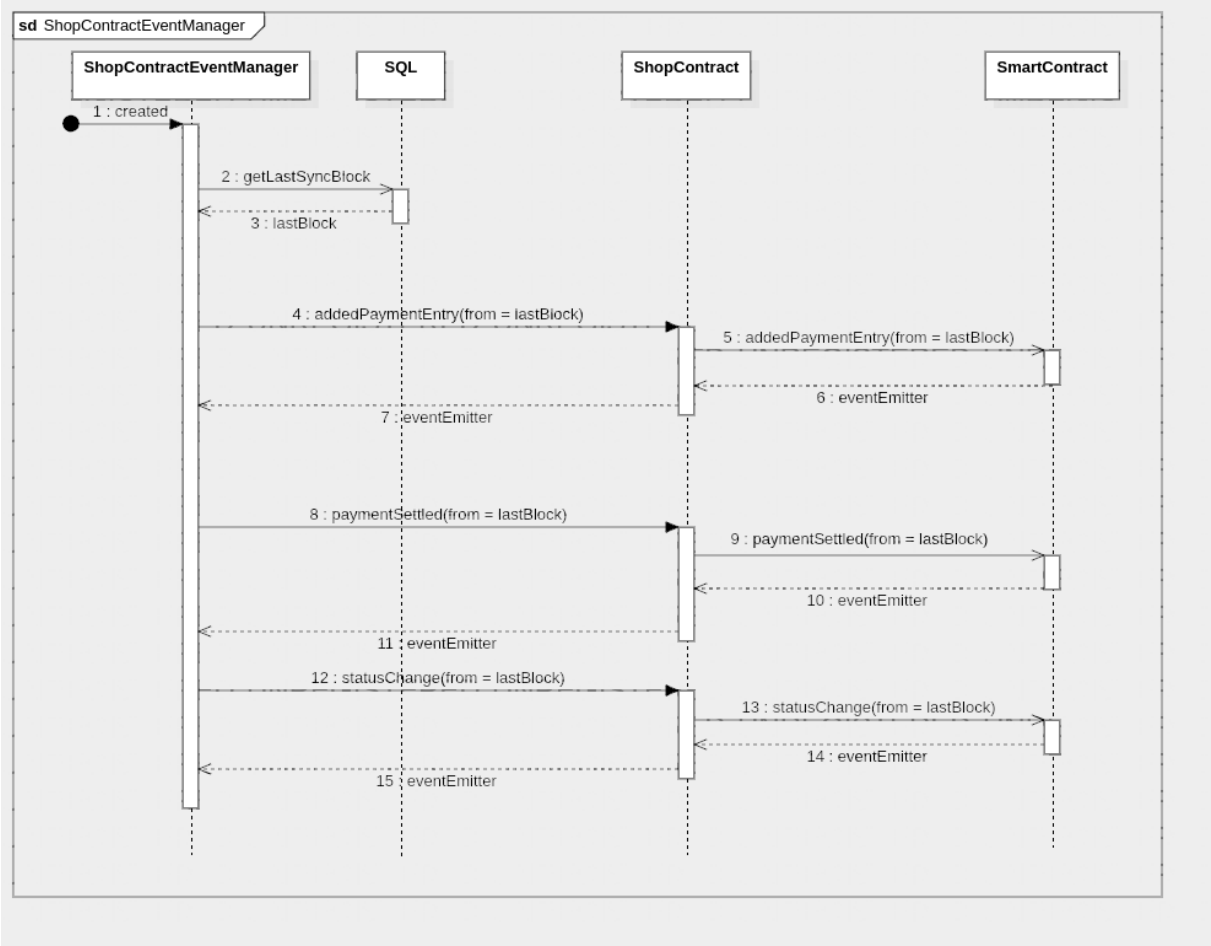

Figure 5: Diagramma di sequenza dell'ascolto degli eventi

Commenti

Mostra la sottoscrizione degli eventi del contratto.

6.3 Nuovo oggetto in vendita

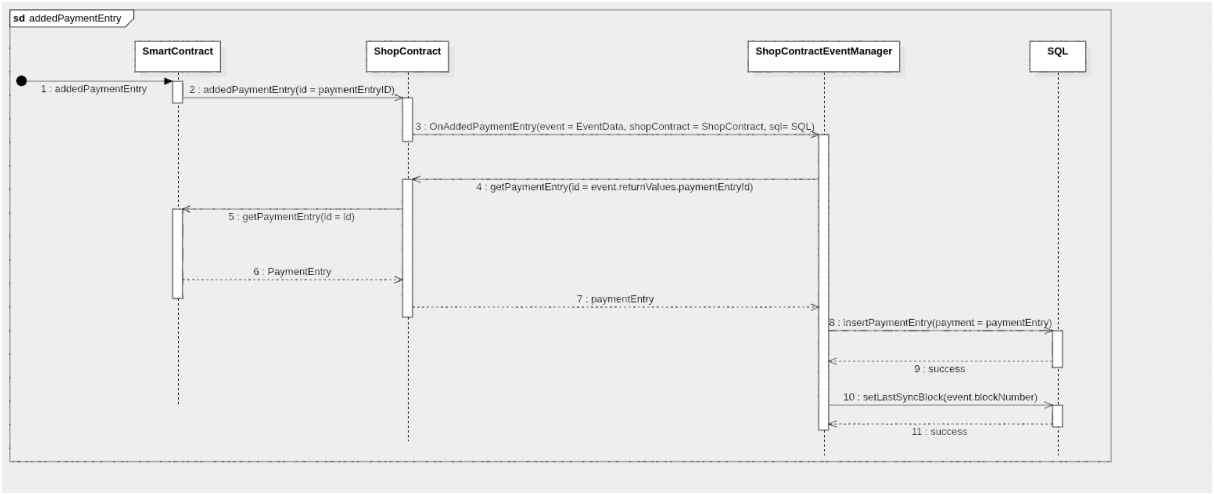

Figure 6: Diagramma di sequenza di un nuovo oggetto in vendita

Commenti

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

6.4 Nuova transazione

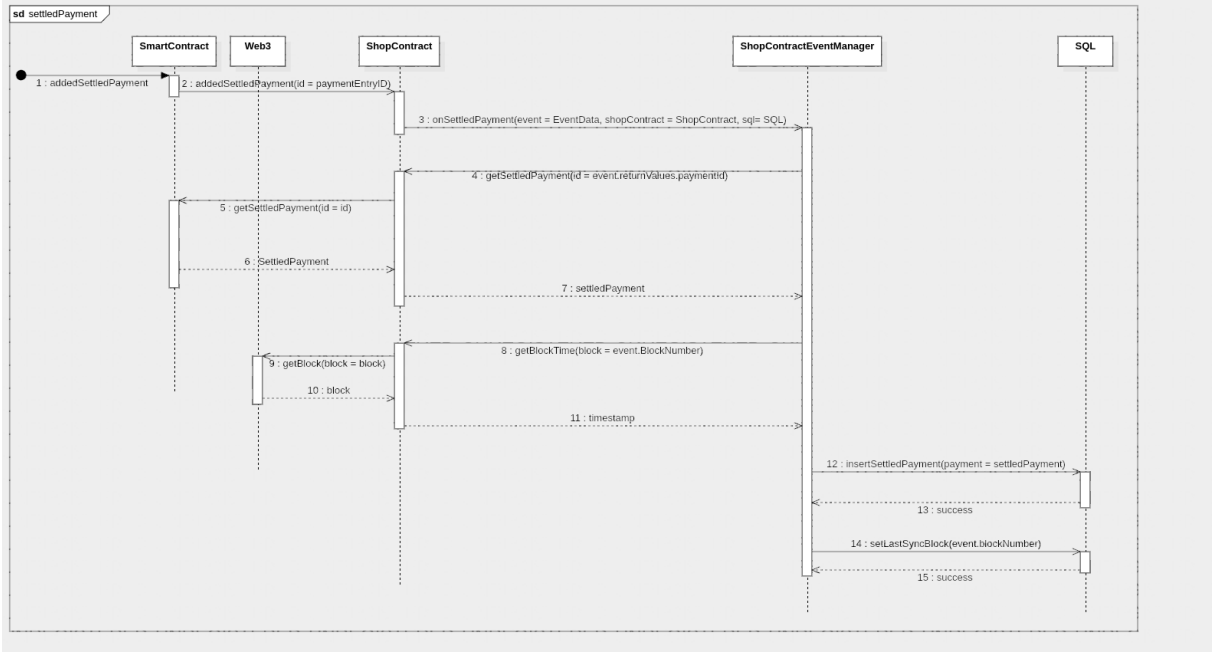

Figure 7: Diagramma di sequenza di una nuova transazione

Commenti

Mostra cosa succede quando viene create una nuova transazione.

6.5 Cambio di stato di una transazione

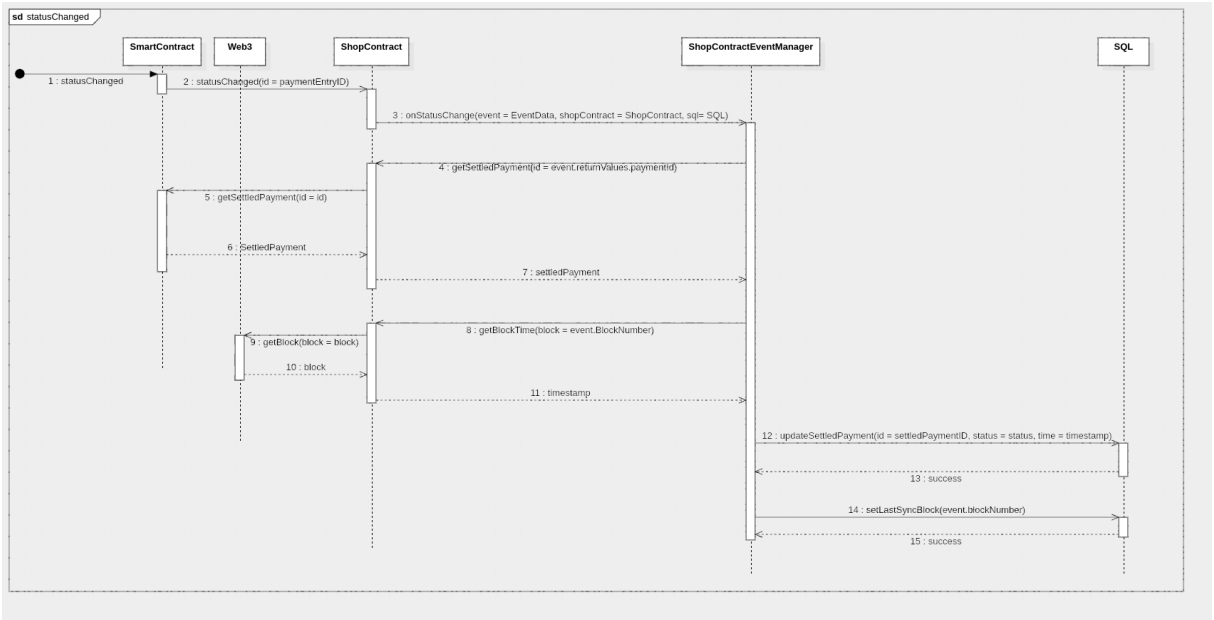

Figure 8: Diagramma di sequenza del cambio di stato di una transazione

Commenti

Mostra cosa succede quando una transazione cambia di stato.

6.6 Pagina transazioni in entrata

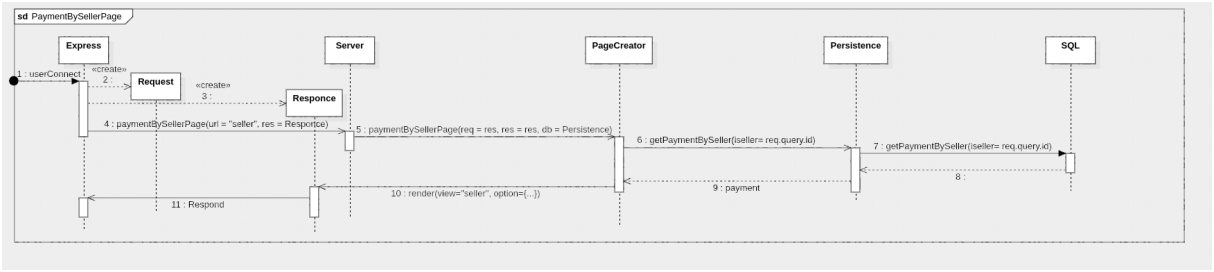

Figure 9: Diagramma di sequenza della richeista della pagina delle transazioni in entrata

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.