POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Smart Contract nell'organizzazione di eventi
Distributed Ledger Layer

Relatori

Prof. Luca ARDITO

Prof. Maurizio MORISIO

Candidato

Pietro CILLUFFO

Aprile 2021

Sommario

***Contesto*:** Le piattoforme digitali di supporto all'organizzazione di eventi non sono molto diffuse, ma allo stesso tempo l'organizzazione di eventi di ogni tipologia come ad esempio l'organizzazione di fiere, di concerti, di matrimoni, meeting aziendali, compleanni è un mercato in grande crescita. Le aziende sono sempre più disposte ad investire, i fornitori ricercano sempre più visibilità nel web e i privati basano sempre di più la ricerca di quali servizi ottenere nel mondo delle piattaforme digitali. DME situa in questo contesto, si vuole ottenere una piattaforma innovativa che coinvolga tutti questi attori e che sia utile per l'organizzazione di un evento cercando di personalizzare il più possibile il servizio.

***Obiettivi*:** L'obiettivo del progetto Digital Managment Events è dunque l'implementazione di questa piattaforma web, la quale è basata su un'architettura a microservizi, il quale è il pattern architetturale che sta avendo il più grande utilizzo per la sua flessibilità. Realizzare un distributed ledger per la definizione e gestione di smart contract come nucleo per la gestione della relazione cliente fornitore durante tutta la durata dell'evento.

***Metodo*:** L'architettura si baserà su 3 layer principali: un layer Front End sviluppato in typescript utilizzando il framework di tendenza, ovvero Angular. Questo layer si occupa di fornire all'utente utilizzatore un'interfaccia il più possibile snella e di facile interpretazione in generale fornisce una user-experience funzionale alle esigenze del cliente; un secondo layer che è il Back End per la produzione dei servizi sviluppato in Java con l'ausilio del framework Spring. I moduli fondamentali sono quelli per la gestione degliaccount e identità per autenticazione e autorizzazione all'uso dei servizi, modulo digestione degli eventi. Ogni modulo offre dei microservizi, che comunicano tra loro con protocolli sincroni o asincroni. Infine una delle innovazioni più importanti che è l layer di Distributed Ledger/smart contract che si occupa di salvare le transazioni tra clienti e fornitori all'interno di un ledger distribuito e di gestire i contratti in modo digitale dematerializzandoli completamente garantendo privacy e riservatezza.

***Conclusioni*:** La piattaforma fornisce agli attori fornitore e utente che ne vorrebbero usufruire le principali funzionalità che si è pensato potrebbero essere utili. Il servizio di gestione dei contratti con il distributed ledger è stato legato ai servizi disponibili raggiungibili dal front end esponendo le chiamate per la modifica della chaincode in un microservizio ad hoc.
Il progetto di tesi ha portato alla realizzazione di buonissima parte dei requisiti richiesti da DME. La piattaforma può essere facilmente modificata per essere adattata ad un utilizzo commerciale, sia da un punto di vista della user experience sia per l'aggiunta di ulteriori funzionalità che inizialmente potrebbero non essere state pensate. Anche la chiancode fornisce le funzioni base per la sua gestione degli smart contracts sia lato fornitore che utente, ma è lasciata aperta ad ulteriori espansioni possibili.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Digital Management of Events è un progetto finanziato dalla regione Puglia, atto a realizzare una piattaforma digitale, della quale è prevista inizialmente un utilizzo nel mercato del territorio pugliese ma con l'obiettivo di espandersi nel territorio nazionale. La piattaforma digitale è rivolta al supporto ad utenti e fornitori che vogliono l'organizzare, gestire, comunicare e partecipare ad eventi di variegate tipologie. Stabilita la vastità del progetto i partner dello stesso sono molti: Il Politecnico di Torino, l'Universtà di Bari, DS Graphic Engineering S.r.l, STRADE S.r.l, Linear System.
In particolare il ruolo che si assume il Politecnico di Torino è quello della Ricerca e Sviluppo della piattaforma in stretta collaborazione con l'azienda Linear System e ancora nel dettaglio l'azienda è divisa in vari membri del team così come noi del Politecnico abbiamo collaborato e diviso il lavoro in tre membri di sviluppo: il sottoscritto Pietro Cilluffo con Giulia Melletti e Martino Massa.
Il processo di sviluppo si è attenuto il più possibile a quelle regole di riferimento della metodologia Agile, in modo da portare avanti il lavoro prettamente a distanza nella maniera più efficace possibile sia tra noi tesisti del Politecnico sia con l'azienda Linear System. Il lavoro del Politecnico previsto durante il periodo che concerne il lavoro di tesi riguarda due macro-periodi che possiamo suddividere in:

- Una parte molto corposa sullo sviluppo del Front End e aggancio alle funzionalità del Back End implementate contemporaneamente da Linear;

- Studio, e analisi delle soluzioni adatte al progetto, basate su Hyperledger per l'implementazione degli Smart Contract;

1.1 Scenario di riferimento: mercato relativo all'organizzazione di eventi

Il settore ha visto una grande crescita nell'ultimo decennio: l'organizzazione di eventi si è affermata sempre più, non solo in quanto attività fine a sé stessa, ma anche come leva determinante nell'ambito del marketing e della comunicazione aziendale. Ad esempio, è in costante aumento la richiesta di professionisti nell'ambito degli eventi da parte delle imprese, oltre che per promuovere prodotti e servizi, anche per organizzare eventi aziendali di team building, volti a motivare e unire il proprio staff. Questo discorso non vale solo per eventi aziendali (per convention, team building, incentivi, lanci di prodotto, ecc.) ma anche per feste private (matrimoni, compleanni, lauree..) oppure per eventi organizzati dalle Pubbliche Amministrazioni (concerti, manifestazioni, eventi sociali, raccolte fondi...).
Sempre nel contesto dell'evento aziendale, ma se vogliamo anche per tutto il pool di tipologie di eventi, l'evento è parte di una strategia di comunicazione e viene utilizzato insieme ad altre forme di comunicazione come per esempio quella digitale (social network in primis), che ne moltiplica l'audience e ne allunga la vita, con un prima e un dopo. In questo senso, l'evento è sempre meno tattico e sempre più strategico, e la sua portata non si esaurisce più nel tempo dello svolgimento. Anche per il mercato italiano naturalmente si registra la trasformazione degli eventi in Live Communication: gli eventi diventano sempre più elementi integranti della communication aziendale, costituiscono la miglior forma di brand experience, sono sempre più live, grazie alle tecnologie digitali e hanno una componente di intrattenimento sempre più rilevante.

Oltre l'intrattenimento che è una componente fondamentale dell'evento, la digitalizzazione sta diventando una componente fondamentale non solo per rendere l'esperienza da parte dell'organizzatore e/o del partecipante unica ma anche grazie alla raccolta di dati, renderlo un'esperienza unica ed ad hoc per ogni singolo organizzatore/partecipante.

Secondo alcuni studi riportati nell'analisi fatta nel documento del progetto DME: "Solo per conferenze aziendali ed eventi per la promozione delle aziende, ogni anno vengono spesi a livello globale circa 512 Miliardi di dollari, mentre il valore complessivo di mercato di settore supera, annualmente, i 3.000 miliardi di dollari. Sempre nel settore degli eventi aziendali in Italia si è registrata una crescita, dopo la crisi del 2008, a partire dal 2014. Gli investimenti delle aziende in eventi sono cresciuti raggiungendo un volume di spesa complessivo pari a 819 milioni di euro nel 2016 e 852 milioni nel 2017. Le previsioni per i prossimi anni sono di crescita costante stimando che per il 2020 gli investimenti in eventi supereranno il miliardo di euro. Gli eventi hanno una propria autonomia, anche di budget. Il trend è

chiaro, in 11 anni di monitoraggio è costantemente aumentata la percentuale di aziende che hanno investito in eventi, a conferma dell'ormai avvenuta integrazione del mezzo nelle strategie di comunicazione. E che gli eventi si siano guadagnati una propria autonomia e un proprio budget è testimoniato anche dal fatto che sempre meno aziende tolgono risorse ad altri mezzi per organizzarli. La metà delle aziende che quest'anno anno hanno organizzato eventi ha mantenuto stabile il budget a fronte di un 29% che lo ha aumentato e di un 20% che lo ha ridotto. L'81% delle aziende prevede di incrementare il budget in eventi nei prossimi due anni."

È evidente che il mercato in questo ambito è molto espansione, dunque una piattaforma come DME potrebbe avere gran successo sia per organizzatori, invitati ma anche per i fornitori che vogliono investire e darsi visibilità.

1.2 Funzionalità ricercate

Il progetto di ricerca DME si pone come obiettivo l'implementazione di un nuovo modello di business e di servizio nel mercato dell'organizzazione degli eventi, innovando radicalmente rispetto agli strumenti ora disponibili, intesi come: marketplace e strumenti di pianificazione.
Le funzionalità principali di DME sono:

- scelta della tipologia di evento (ad es matrimonio, conferenza, compleanno..)
- in base alla tipologia, proposta di un modello organizzativo specifico, in termini di tempistiche, beni e servizi più ricorrenti.
- marketplace di beni e servizi per tipo di evento per il confronto e la scelta dei fornitori
- contrattualistica standard e supporto alla sua definizione, gestione e controllo per l'interazione con i fornitori
- controllo di avanzamento delle attività
- contabilità, con budget preventivo e consuntivo, tracciabilità di forniture e servizi
- interazione con i fruitori dell'evento (da emissione di eventuale biglietto a condivisione di dati e programmi, condivisione di foto e video) su vari canali (app dedicata, social network). Il tutto dovrà essere caratterizzato da una user experience di alto livello, in modalità multicanale (web e mobile).

Capitolo 2

Architettura Microservizi

2.1 Introduzione

Un'architettura di microservizi è costituita da un insieme di servizi ridotti e autonomi. Ogni servizio è indipendente e deve implementare una singola funzionalità di business. Nelle architetture monolitiche tutti i processi sono strettamente collegati

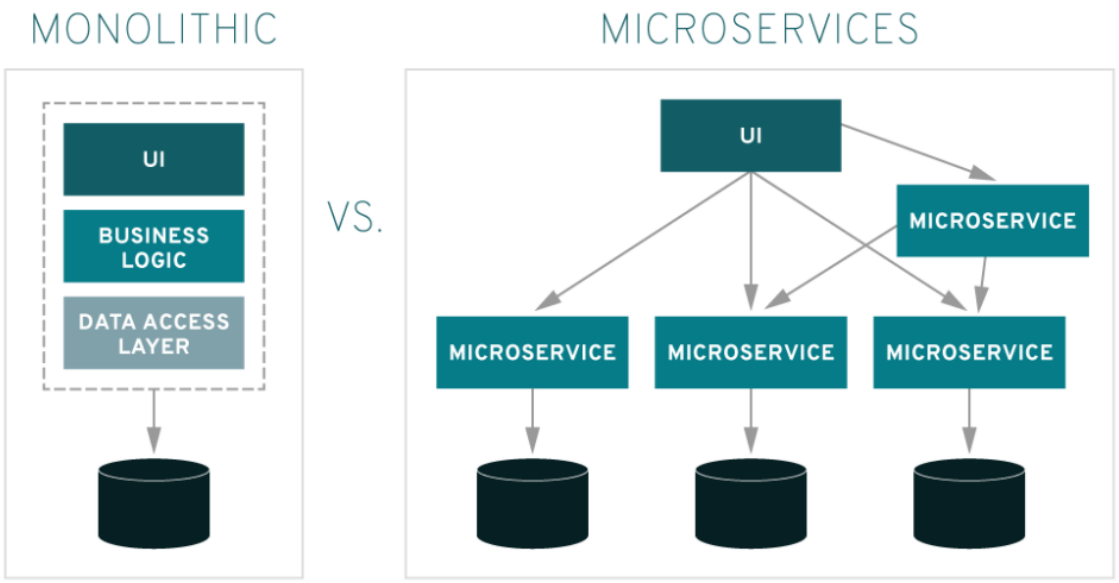


Figura 2.1: confronto tra schema monolitico e schema microservizi[1]

tra loro e vengono eseguiti come un singolo servizio. Ciò significa che se un processo dell'applicazione sperimenta un picco nella richiesta, è necessario ridimensionare l'intera architettura. Aggiungere o migliorare una funzionalità dell'applicazione

monolitica diventa più complesso, in quanto sarà necessario aumentare la base di codice. Tale complessità limita la sperimentazione e rende più difficile implementare nuove idee. Le architetture monolitiche rappresentano un ulteriore rischio per la disponibilità dell'applicazione, poiché la presenza di numerosi processi dipendenti e strettamente collegati aumenta l'impatto di un errore in un singolo processo.

Con un'architettura basata su microservizi, un'applicazione è realizzata da componenti indipendenti che eseguono ciascun processo applicativo come un servizio. Tali servizi comunicano attraverso un'interfaccia ben definita che utilizza API leggere. I servizi sono realizzati per le funzioni aziendali e ogni servizio esegue una sola funzione. Poiché eseguito in modo indipendente, ciascun servizio può essere aggiornato, distribuito e ridimensionato per rispondere alla richiesta di funzioni specifiche di un'applicazione. Le dimensioni dei microservizi sono tali da consentirne la scrittura e la gestione da parte di un unico piccolo team di sviluppatori. Un team può aggiornare un servizio esistente senza ricompilare e ridistribuire l'intera applicazione.

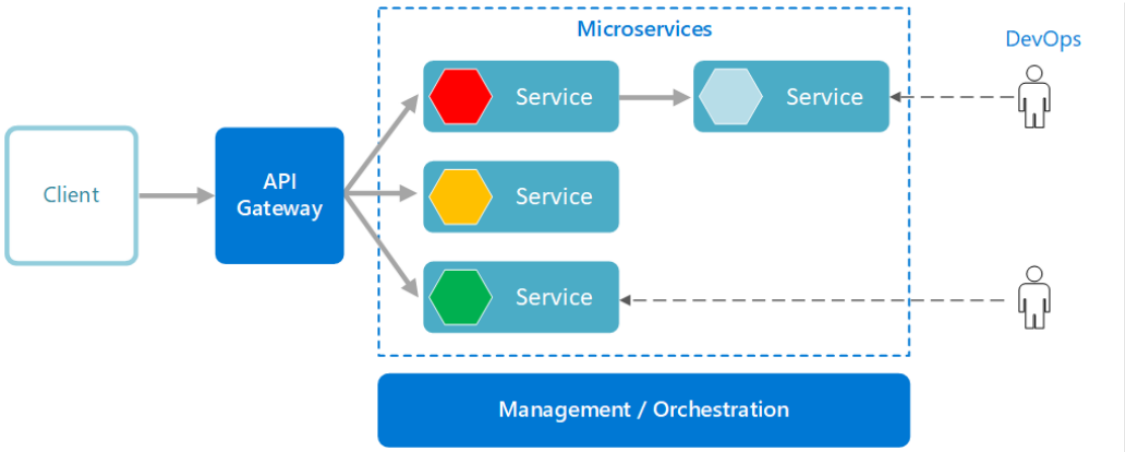


Figura 2.2: schema microservizio[2]

I servizi sono responsabili della persistenza dei propri dati o dello stato esterno. Questo comportamento differisce dal modello tradizionale, in cui la persistenza dei dati viene gestita da un livello dati distinto. I servizi comunicano tra loro tramite API ben definite. I dettagli di implementazione interna di ogni servizio sono nascosti da altri servizi. Non è necessario che i servizi condividano lo stesso stack di tecnologie, le stesse librerie o gli stessi framework. Oltre ai servizi, in un'architettura di microservizi tipica compaiono alcuni altri componenti:

- Gestione/orchestrazione: Questo componente è responsabile del posizionamento dei servizi sui nodi, dell'identificazione degli errori, del ribilanciamento dei servizi tra i nodi e così via.

- Gateway API: Il gateway API è il punto di ingresso per i client. Invece di chiamare direttamente i servizi, i client chiamano il gateway API, che inoltra la chiamata ai servizi appropriati sul back-end.

  I vantaggi associati all'uso di un gateway API includono la separazione dei client dai servizi, di cui può essere effettuato il refactoring o il controllo delle versioni senza la necessità di aggiornare tutti i client, un altro vantaggio è l'utilizzo da parte dei servizi di protocolli di messaggistica non compatibili con il Web, come AMQP, infine il gateway API può eseguire altre funzioni trasversali, come l'autenticazione, la registrazione, la terminazione SSL e il bilanciamento del carico.

[2]

Suddividere un'app nelle relative funzioni base ed evitare le insidie dell'approccio monolitico possono sembrare concetti familiari, perché lo stile architetturale dei microservizi è simile a quello dell'architettura SOA (Service-Oriented Architecture), uno stile di progettazione del software ormai consolidato.

Ai primordi dello sviluppo applicativo, anche un cambiamento minimo a un'app esistente imponeva un aggiornamento completo e un ciclo di controllo qualità (QA, Quality Assurance) a sé, che rischiava di rallentare il lavoro di vari team secondari. Tale approccio viene spesso definito "monolitico", perché il codice sorgente dell'intera app veniva compilato in una singola unità di deployment (ad esempio con estensione .war o .ear). Se gli aggiornamenti a una parte dell'app provocavano errori, era necessario disconnettere tutto, fare un passo indietro e correggere. Questo approccio è ancora applicabile alle piccole applicazioni, ma le aziende in crescita non possono permettersi tempi di inattività.

E qui entra in gioco l'architettura SOA (Service-Oriented Architecture), in cui le app sono strutturate in servizi riutilizzabili che comunicano fra loro tramite un Enterprise Service Bus (ESB). In questa architettura i singoli servizi sono incentrati su uno specifico processo aziendale e seguono un protocollo di comunicazione, tra cui SOAP, ActiveMQ o Apache Thrift, per essere condivisi attraverso l'ESB. Nel complesso, questa suite di servizi integrati attraverso un ESB costituisce un'applicazione.

Inoltre, questo metodo permette di compilare, testare e modificare più servizi contemporaneamente, svincolando i team IT dai cicli di sviluppo monolitici. Tuttavia, poiché l'ESB rappresenta un singolo punto di errore per l'intero sistema, potrebbe comportare un ostacolo per l'intera organizzazione.

La differenza tra architettura SOA e i microservizi è che microservizi possono comunicare tra loro, in genere in modalità stateless, permettendo di realizzare app con una maggiore tolleranza di errore e meno dipendenti da un singolo ESB (enterprise service bus). Inoltre, comunicano tramite interfacce di programmazione

delle applicazioni (API) indipendenti dal linguaggio e ciò consente ad ogni team di sviluppo di scegliere i propri strumenti.

Considerando l'evoluzione di SOA, i microservizi non costituiscono una novità assoluta, ma ultimamente sono diventati più appetibili grazie ai progressi delle tecnologie di containerizzazione. Oggi i container permettono di eseguire più parti di un'app in modo indipendente, sullo stesso hardware, con un controllo decisamente superiore sui singoli componenti e cicli di vita. I microservizi containerizzati rappresentano la base delle applicazioni cloud-native.

2.2 I vantaggi di un'architettura basata sui microservizi

Basandosi su un'architettura distribuita, i microservizi consentono di ottenere sviluppo e routine più efficienti. La possibilità di sviluppare più microservizi in contemporanea consente a più sviluppatori di lavorare simultaneamente alla stessa app, riducendo le tempistiche di sviluppo.

- **Time-to-market più rapido** Consentendo di abbreviare i cicli di sviluppo, un'architettura basata su microservizi supporta deployment e aggiornamenti più agili.

- **Scalabilità superiore** Man mano che la domanda per determinati servizi aumenta, i microservizi possono essere distribuiti su più server e infrastrutture, in base alle esigenze aziendali.

- **Resilienza** Ciascun servizio, se costruito correttamente, è indipendente e non influisce sugli altri servizi nell'infrastruttura. Di conseguenza, l'eventuale errore di un componente non determina il blocco dell'intera app, come avviene con il modello monolitico.

- **Semplicità di deployment** Poiché le app basate su microservizi sono più piccole e modulari delle tradizionali applicazioni monolitiche, tutti i problemi associati a tali deployment vengono automaticamente eliminati. Benché questo approccio richieda un coordinamento superiore, i vantaggi che ne derivano sono determinanti.

- **Accessibilità** Poiché le app più grandi vengono suddivise in parti più piccole, per gli sviluppatori è molto più semplice comprendere, aggiornare e migliorare tali componenti, e questo permette di accelerare i cicli di sviluppo, soprattutto in combinazione con le metodologie di sviluppo Agile.

- **Maggiore apertura** Grazie alle API indipendenti dal linguaggio, gli sviluppatori sono liberi di scegliere il linguaggio e la tecnologia ottimali per la funzione da creare.

[1]

2.3 Problematiche

Un'architettura basata su microservizi presenta due criticità principali: la complessità e la produttività. Nel suo intervento presso il Red Hat Summit del 2017, il platform architect di Red Hat Mobile John Frizelle ha identificato queste otto categorie di problematiche:

- Compilazione: occorre dedicare tempo all'identificazione delle dipendenze fra i servizi. A causa di tali dipendenze, quando si esegue una compilazione può essere necessario eseguirne anche molte altre. È fondamentale considerare anche gli effetti prodotti dai microservizi sui tuoi dati.

- Test: i test di integrazione, così come i test end-to-end, possono diventare molto difficili, ma anche più importanti che mai. Occorre ricordarsi che un errore in una parte dell'architettura potrebbe causare un errore in un componente a vari passi di distanza, a seconda del modo in cui i servizi sono strutturati per supportarsi a vicenda.

- Gestione delle versioni: l'aggiornamento a una nuova versione potrebbe compromettere la retrocompatibilità. Il problema può essere gestito attraverso la logica condizionale, ma in breve tempo può diventare difficile da gestire. Disporre di più versioni live per client diversi può essere un'alternativa, ma la manutenzione e la gestione possono essere molto più laboriose.

- Deployment: durante la configurazione iniziale, anche questa è una fase critica. Per semplificare il deployment, occorre innanzitutto investire notevolmente in soluzioni di automazione. La complessità dei microservizi renderebbe il deployment manuale estremamente difficile. Pensa al modo e all'ordine in cui eseguire il roll-out dei servizi.

- Registrazione: nei sistemi distribuiti, per ricollegare tutti i vari componenti sono necessari registri centralizzati, senza i quali gestirli in modo scalabile risulterebbe impossibile.

- Monitoraggio: è fondamentale per i team operativi avere una visibilità centralizzata del sistema, al fine di identificare l'origine dei problemi.

- Debugging: il debugging remoto non è una scelta praticabile con decine o centinaia di servizi. Al momento non esiste un'unica soluzione legata al debugging.
- Connettività: occorre valutare quale metodo di rilevamento dei servizi scegliere, se centralizzato o integrato.

[1]

Capitolo 3

Architettura DME

3.1 Architettura

Digital Management Event è una piattaforma visibile dall'utente come un insieme di servizi fruibili attraverso l'utilizzo del Web.
La sua architettura può essere descritta come nell'immagine seguente:

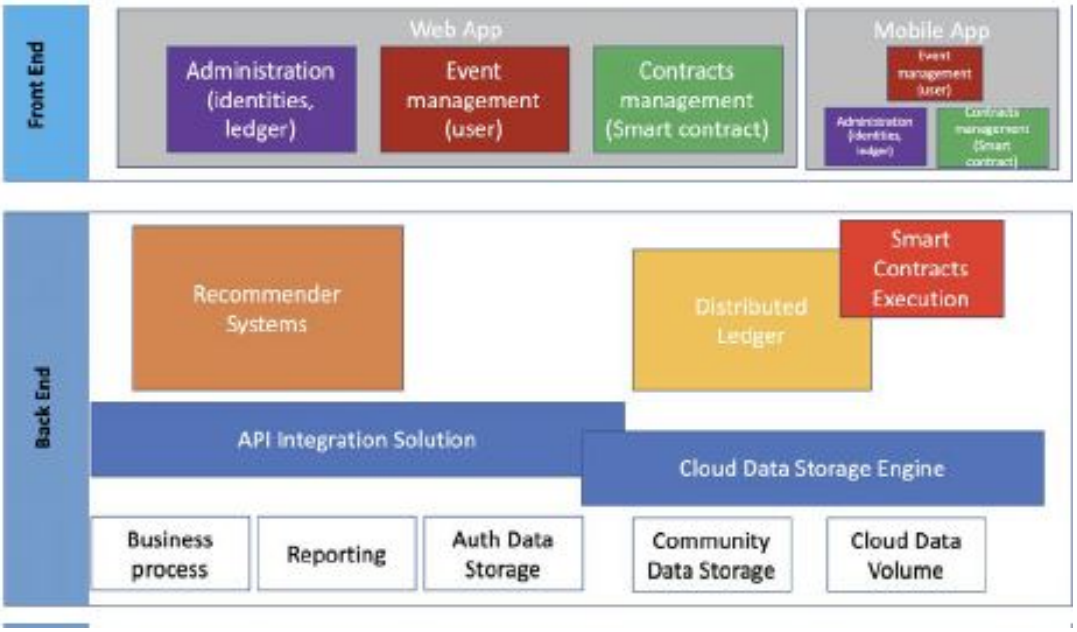


Figura 3.1: Schema a blocchi della piattaforma DME

Come si evince dall'immagine i layer principali sono due:

- Layer di Front End: responsabile di fornire un'interfaccia utente user-friendly per l'accesso ai vari servizi della piattaforma. L'interfaccia è il più possibile responsive alle varie tipologie di device: dalla versione desktop, al tablet, al comune smartphone.
  Questo livello ha contemporaneamente il compito di comunicare con i microservizi di Back End per recuperare i dati necessari e mostrarli all'utente utilizzatore;

- Layer di Back End: responsabile di dialogare con lo strato di persistenza e fornire le risposte al Front End utilizzando una business logic adatta e ottimizzata. Il Back End permetterà tutto questo mettendo a disposizione delle API REST contattabili dal layer di Front End, in maniera sicura tramite l' utilizzo del protocollo di autenticazione OAuth2.
  All'interno dello stesso layer vi è un modulo di distributed ledger per implementare gli smart contract che è messo a disposizione mediante un servizio che espone delle API apposite, al fine di trattare le transazioni e i pagamenti tra fornitore e utente;

Ogni modulo offre dei microservizi, che comunicano tra loro con protocolli sincroni o asincroni.
Gli elementi di riferimento per la progettazione sono i seguenti:

- Creazione di un sistema di comunicazione tra i microservizi, attraverso le componenti di Business Logic, usando un protocollo di comunicazione tra processi. In linea generale si è utilizzato il protocollo sincrono HTTP/HTTPS per le chiamate ai servizi Web API mentre la comunicazione tra componenti di Business Logic (a valle dell'invocazione di uno o più microservizi) avviene in maniera asincrona per garantire l'autonomia dei singoli microservizi;

- Realizzazione delle API che consentono la collaborazione tra risorse interne ed esterne alla piattaforma tenendo conto di ciò che verrà mostrato all'utente;

- Progettazione di un distributed ledger che consente la creazione e l'esecuzione di smart contract tra clienti e fornitori.

- Realizzazione di opportuni mockup grafici per stabilire gli aspetti di rilevanza per l'esperienza visiva di insieme

3.1.1 Obiettivi e vincoli

L'organizzazione di eventi di ogni tipo (concerti, fiere, matrimoni, meeting aziendali, compleanni, ecc) é un mercato in grande crescita, che coinvolge molti attori e stimola a sua volta diverse attività economiche di supporto.

Il progetto DME si situa in questo contesto, e si propone di realizzare una piattaforma digitale di supporto alla organizzazione, gestione, comunicazione e partecipazione ad eventi.
Per raggiungere i risultati sopra descritti, il progetto si pone questi obiettivi:

- Realizzare una piattaforma, basata su architettura a microservizi (MA, Microservice Architecture), di funzionalità elementari riusabili a supporto di definizione di eventi e tipi di eventi, realizzazione di marketplace di definizione di beni e servizi, definizione di utenti e profili, servizi orizzontali di autorizzazione e autenticazione, backup e recovery.

- Realizzare un distributed ledger per la definizione e gestione di smart contract come nucleo per la gestione della relazione cliente fornitore durante tutta la durata dell'evento

- Realizzare servizi per la raccolta, analisi e modellazione di dati su eventi,beni, servizi, clienti. L'analisi intelligente di questi big data con una varietà di algoritmi di Machine learning per costruire profili ricorrenti di clienti ed eventi è propedeutica al punto seguente

- Realizzare recommender system per fornire a clienti vecchi e nuovi proposte di eventi tipo (comprendenti modello organizzativo, pacchetti di beni e servizi, modalità di collaborazione e interazione) in funzione del profilo del cliente capaci di semplificare al massimo la complessità e difficoltà insite nell'organizzazione e gestione di eventi, sia per utenti privati che professionali

- Realizzare componenti per la fornitura multicanale (web, mobile, social networks) dei servizi e processi visti sopra, con particolare enfasi su una user experience di alto livello, in linea con le migliori offerte attualmente sul mercato.

3.1.2 Casi d'uso

Figura 3.2: Casi d'uso

Una descrizione più accurata dei singoli casi d'uso è presente al capitolo 4.3.4

3.1.3 Panoramica logica

Qui si illustra una lista d'insieme di tutte le API esposte dai microservizi:

Security_repo:

- */oauth/token?grant_type=password&username=utente&password=password*
  Metodo POST. Contiene un header Authorization con clientId e clientSecret, restituisce il jwtToken corrispondente all'utente e password inseriti nell'url.
- */oauth/revoke-token*
  Metodo GET. Contiene un header Authorization con un jwtToken. Invalid ail token per fare logout dell'utente
- */user/update_password?newPassword=nuovapass*
  Metodo POST. Contiene un header Authorization con un jwtToken. Cambia la password dell'utente associate al token con quella inserita nell'url

DME_Utenti:

- /utente/createUser
  Metodo POST. Crea un nuovo utente sul db con i dati contenuti nel body
- /utente/findUserById
  Metodo POST. Contiene nel body l'IdUtente. Restituisce l'utente associato a quell'id

DME_Clienti:

- /clienti/createCliente
  Metodo POST. Crea un nuovo cliente sul db con i dati contenuti nel body. Deve riferirsi ad un idUtente già esistente
- /clienti/findClienteByUserId
  Metodo POST. Contiene nel body l'IdUtente. Restituisce il cliente associato a quell'id
- /clienti/updateCliente
  Metodo PUT. Contiene nel body i nuovi dati del cliente e l'id associato. Sostituisce i dati del cliente con quelli nuovi
- /ordini/createOrdine
  Metodo POST. Crea l'entità ordine con i dati contenuti nel body

- /ordini/deleteOrdineByIdOrdine
  Metodo DELETE. Elimina l'ordine associato all'idOrdine contenuto nel body
- /ordini/findOrdineByIdOrdine
  Metodo POST. Restituisce l'ordine associato all'idOrdine contenuto nel body
- /ordini/findOrdiniByIdCliente
  Metodo POST. Restituisce gli ordini associati all'idCliente contenuto nel body
- /ordini/findOrdiniByIdFornitore
  Metodo POST. Restituisce gli ordini associati all'idFornitore contenuto nel body
- /ordini/updateOrdine
  Metodo PUT. Aggiorna l'entità ordine contenuta nel body

DME_Fornitori:

- /fornitori/createFornitore
  Metodo POST. Crea un nuovo fornitore sul db con i dati contenuti nel body. Deve riferirsi ad un idUtente già esistente
- /fornitori/findFornitoreByUserId
  Metodo POST. Contiene nel body l'IdUtente. Restituisce il fornitore associato a quell'id
- /fornitori/updateFornitore
  Metodo PUT. Contiene nel body i nuovi dati del fornitore e l'id associato. Sostituisce i dati del fornitore con quelli nuovi
- /listini/createListini
  Metodo POST. Crea una nuova entità listini sul db con i dati contenuti nel body
- /listini/findListiniByIdListino
  Metodo POST. Restituisce l'entità listino associata all'idListino contenuto nel body
- /tipoPagAccettato/createTipoPagAccettato
  Metodo POST. Crea una nuova entità TipoPagAccettato con i dati contenuti nel body, l'entità è associata al singolo fornitore
- /tipoPagAccettato/findTipoPagAccettatoByIdFornitore
  Metodo POST. Restituisce le entità TipoPagAccettato associate all'idFornitore contenuto nel body

- /tipoPagAccettato/deleteTipoPagAccettatoByIdFornitore
  Metodo DELETE. Elimina le entità TipoPagAccettato associate all'idFornitore contenuto nel body

- /modPagAccettato/createModPagAccettato
  Metodo POST. Crea una nuova entità ModPagAccettato con i dati contenuti nel body, l'entità è associata al singolo fornitore

- /modPagAccettato/findModPagAccettatoByIdFornitore
  Metodo POST. Restituisce le entità ModPagAccettato associate all'idFornitore contenuto nel body

- /puntiVendita/createPuntoVendita
  Metodo POST. Crea l'entità puntoVendita con i dati contenuti nel body, l'entità è associata ad un fornitore

- /puntiVendita/findPuntiVenditaByIdFornitore
  Metodo POST. Restituisce le entità puntoVendita associate all'idFornitore contenuto nel body

- /puntiVendita/updatePuntoVendita
  Metodo PUT. Aggiorna l'entità puntoVendita contenuta nel body

- /orari/createOrari
  Metodo POST. Crea l'entità orari con i dati contenuti nel body, l'entità è associata ad un punto vendita

- /orari/findOrariByIdPuntoVendita
  Metodo POST. Restituisce l'entità orari associata all'idPuntoVendita contenuto nel body

- /orari/updateOrari
  Metodo PUT. Aggiorna l'entità orari contenuta nel body

DME_Anagrafiche:

- /anagrafiche/findAllProfessioni
  Metodo GET. Restituisce tutti i tipi di professione presenti nel db

- /anagrafiche/findAllTipiPagamento
  Metodo GET. Restituisce tutti i tipi di pagamento presenti nel db

- /anagrafiche/findAllModalitaPagamento
  Metodo GET. Restituisce tutte le modalità di pagamento presenti nel db

- /anagrafiche/findAllMansioni
  Metodo GET. Restituisce tutte le mansioni presenti nel db
- /anagrafiche/findAllCategorieServizio
  Metodo GET. Restituisce tutte le categorie di servizio presenti nel db
- /anagrafiche/findAllTipiEventi
  Metodo GET. Restituisce tutte i tipi di evento presenti nel db
- /anagrafiche/findAllComuni
  Metodo GET. Restituisce tutte i comuni presenti nel db

DME_Evento:

- /eventi/createEvento
  Metodo POST. Crea un nuovo evento sul db con i dati contenuti nel body
- /eventi/findEventiByIdCliente
  Metodo POST. Restituisce gli eventi associati all'idCliente contenuto nel body
- /eventi/findEventoByIdEvento
  Metodo POST. Restituisce gli eventi associati all'idEvento contenuto nel body
- /eventi/updateEvento
  Metodo PUT. Aggiorna l'evento contenuto nel body
- /invitati/createInvitati
  Metodo POST. Crea un nuovo invitato associato ad un evento con i dati contenuti nel body
- /invitati/updateInvitati
  Metodo PUT. Aggiorna l'invitato contenuto nel body
- /invitati/deleteInvitatiByIdInvitato
  Metodo DELETE. Elimina l'invitato associato all'idInvitato contenuto nel body
- /invitati/findInvitatiByIdInvitato
  Metodo POST. Restiruisce l'invitato associato all'idInvitato contenuto nel body
- /gestioneLuogoInvitati/createGestioneLuogoInvitati
  Metodo POST. Crea un'entità GestioneLuogoInvitati con i dati contenuti nel body, per raggruppare gli invitati in base ad un tavolo o un luogo

- /gestioneLuogoInvitati/updateGestioneLuogoInvitati
  Metodo PUT. Aggiorna l'entità GestioneLuogoInvitati contenuta nel body
- /gestioneLuogoInvitati/findGestioneLuogoInvitatiByIdLuogoInvitati
  Metodo POST. Restituisci l'entità GestioneLuogoInvitati associata all'idLuogoInvitati contenuto nel body
- /gestioneLuogoInvitati/deleteGestioneLuogoInvitatiByIdLuogoInvitati
  Metodo DELETE. Elimina l'entità GestioneLuogoInvitati associata all'idLuogoInvitati contenuto nel body

DME_Agenda:

- /appuntamento/findAllAppuntamenti
  Metodo GET. Restituisce tutte le entità appuntamenti
- /appuntamento/createAppuntamento
  Metodo POST. Crea sul db una nuova entità appuntamento usando i dati contenuti nel body
- /appuntamento/findAppuntamentiByIdFornitoreVetrine
  Metodo POST. Restituisce gli appuntamenti associato all'idFornitoreVetrina inserito nel body
- /appuntamento/findAppuntamentoByIdAppuntamento
  Metodo POST. Restituisce l'appuntamento associato all'idAppuntamento contenuto nel body
- /appuntamento/updateAppuntamento
  Metodo PUT. Aggiorna l'appuntamento contenuto nel body con i nuovi dati inviati
- /clienteAppuntamento/findAllClienteAppuntamenti
  Metodo GET. Restituisce tutte le entità ClienteAppuntamenti, la tabella di relazione tra cliente e appuntamenti
- /clienteAppuntamento/createClienteAppuntamento
  Metodo POST. Crea la nuova entità ClienteAppuntamento contenuta nel body
- /clienteAppuntamento/findClienteAppuntamentiByIdCliente
  Metodo POST. Restituisce tutti i ClienteAppuntamento con l'idCliente contenuto nel body
- /clienteAppuntamento/updateClienteAppuntamento
  Metodo PUT. Aggiorna l'entità ClienteAppuntamneto contenuta nel body

- /fornitoreAppuntamento/findAllFornitoreAppuntamenti
  Metodo GET. Restituisce tutte le entità FornitoreAppuntamenti, la tabella di relazione tra fornitore e appuntamenti

- /fornitoreAppuntamento/createFornitoreAppuntamento
  Metodo POST. Crea la nuova entità FornitoreAppuntamento contenuta nel body

- /fornitoreAppuntamento/findFornitoreAppuntamentiByIdFornitore
  Metodo POST. Restituisce tutti i ClienteAppuntamento con l'idFornitore contenuto nel body

- /fornitoreAppuntamento/updateFornitoreAppuntamento
  Metodo PUT. Aggiorna l'entità FornitoreAppuntamneto contenuta nel body

- /appuntamento/deleteAppuntamentoByIdAppuntamento
  Metodo DELETE. Cancella l'appuntamento associato all'IdAppuntamento contenuto nel body

- /clienteAppuntamento/deleteClienteAppuntamentoByIdAppuntamento
  Metodo DELETE Cancella l'entità ClienteAppuntamento associata all'idCliente Appuntamento contenuto nel body

- /fornitoreAppuntamento/deleteFornitoreAppuntamentoByIdAppuntamento
  Metodo DELETE Cancella l'entità FornitoreAppuntamento associata all'idFornitore Appuntamento contenuto nel body

DME_Vetrina:

- vetrine/createVetrina
  Metodo POST. Crea l'entità vetrina con i dati contenuti nel body, rappresenta un servizio offerto da un fornitore

- vetrine/updateVetrina
  Metodo PUT. Aggiorna l'entità vetrina contenuta nel body

- vetrine/findVetrinaById
  Metodo POST. Restituisce la vetrina associata all'idVetrina contenuto nel body

- vetrine/deleteVetrina
  Metodo DELETE. Elimina la vetrina associata all'idVetrina contenuto nel body

- fornitoriVetrine/createFornitoriVetrina
  Metodo POST. Crea l'entità fornitoreVetrina con i dati contenuti nel body, l'entità rappresenta la relazione tra vetrine e fornitori

- fornitoriVetrine/findFornitoriVetrinaByIdFornitore
  Metodo POST. Restituisce l'entità fornitoriVetrina associata all'idFornitore contenuto nel body

- fornitoriVetrine/findFornitoriVetrinaByCodCategoriaServizio
  Metodo POST. Restituisce l'entità fornitoriVetrina associata al codCategoria-Servizio contenuto nel body

- fornitoriVetrine/findFornitoriVetrinaByIdVetrina
  Metodo POST. Restituisce l'entità fornitoriVetrina associata all'idVetrina contenuto nel body

- fornitoriVetrine/deleteFornitoriVetrina
  Metodo DELETE. Elimina l'entità fornitoreVetrina associata agli idFornitore e IdVetrina contenuti nel body

DME_Contratti:

- /contratti/createContratto
  Metodo POST. Crea un nuovo contratto sul db con i dati contenuti nel body

- /contratti/updateContratto
  Metodo PUT. Aggiorna il contratto sul db con i dati contenuti nel body

- /contratti/findContrattoByIdContratto
  Metodo POST. Restituisce il contratto associato all'idContratto contenuto nel body

- /contratti/findContrattoByIdCliente
  Metodo POST. Restituisce i contratti associati all'idCliente contenuto nel body

- /contratti/findContrattoByIdFornitore
  Metodo POST. Restituisce i contratti associati all'idFornitore contenuto nel body

- /contratti/findContrattoByIdContratto
  Metodo DELETE. Elimina il contratto associato all'idContratto contenuto nel body

Modulo Blockchain:

- /transaction/createTransaction
  Metodo POST. Contiene un header userId con l'id dell'utente registrato nel wallet della chiancode. Crea un nuovo smart-contracts nella chaincode con i dati contenuti nel body
- /transaction/updateTransaction
  Metodo PUT. Contiene un header userId con l'id dell'utente registrato nel wallet della chiancode. Aggiorna lo smart-contracts contenuto nel body
- /transaction/getTransaction
  Metodo POST. Contiene un header userId con l'id dell'utente registrato nel wallet della chiancode. Restituisce lo smart-contract associato all'id contenuto nel body
- /transaction/getTransaction
  Metodo DELETE. Contiene un header userId con l'id dell'utente registrato nel wallet della chiancode. Elimina lo smart-contract associato all'id contenuto nel body
- /transaction/getTransactionHistory
  Metodo POST. Contiene un header userId con l'id dell'utente registrato nel wallet della chiancode. Restituisce la storia delle modifiche dello smart-contract associato all'id contenuto nel body

Microservizio Recommender:
Questo microservizio si interpone tra il front end e i microservizi DME Vetrina, DME Eventi, DME Fornitori e DME Clienti. Per le API in cui si restituisce una lista di entità, il microservizio Recommender elaborerà i dati in base al profilo dell'utente. Le API esposte saranno:

- getCategorieServizioConsigliateByCodTipoEvento
- getVetrineConsigliateByCodCategoriaServizio
- getArticoliSimiliByIdVetrina

3.1.4 Panoramica dei dati

Si riporta la lista dei diagrammi che descrivono la struttura dei dati e le relazioni tra le principali entità all'interno del db per ogni microservizio.

- **Microservizio Utente:**

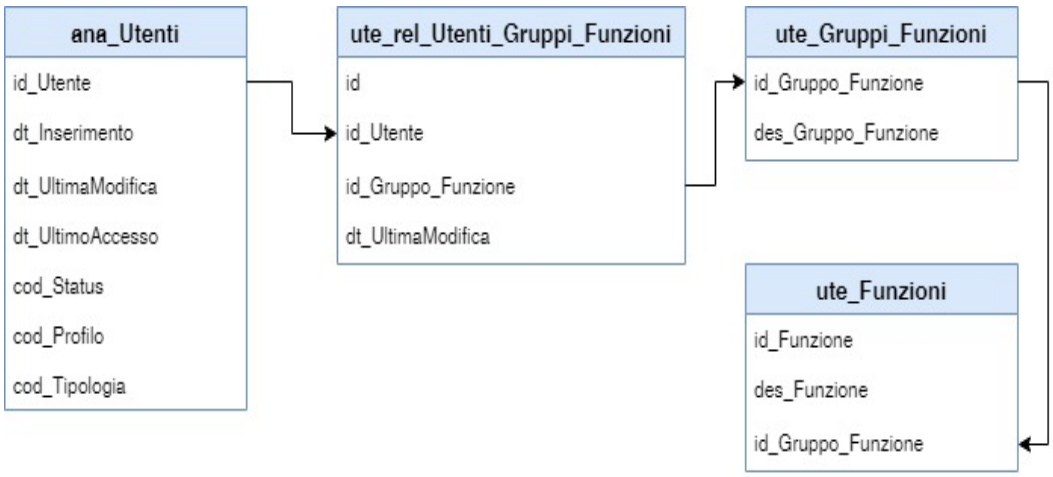


Figura 3.3: Microservizio Utente

Nella tabella ana_Utenti il campo cod_Tipologia identifica se l'utente è: C-Cliente F-Fornitore E-Entrambi. Il campo cod_Profilo identifica F-Persona fisica G-Persona giuridica, cod-Status se è A-Attivo N-Non attivo

- **Microservizio Cliente:**

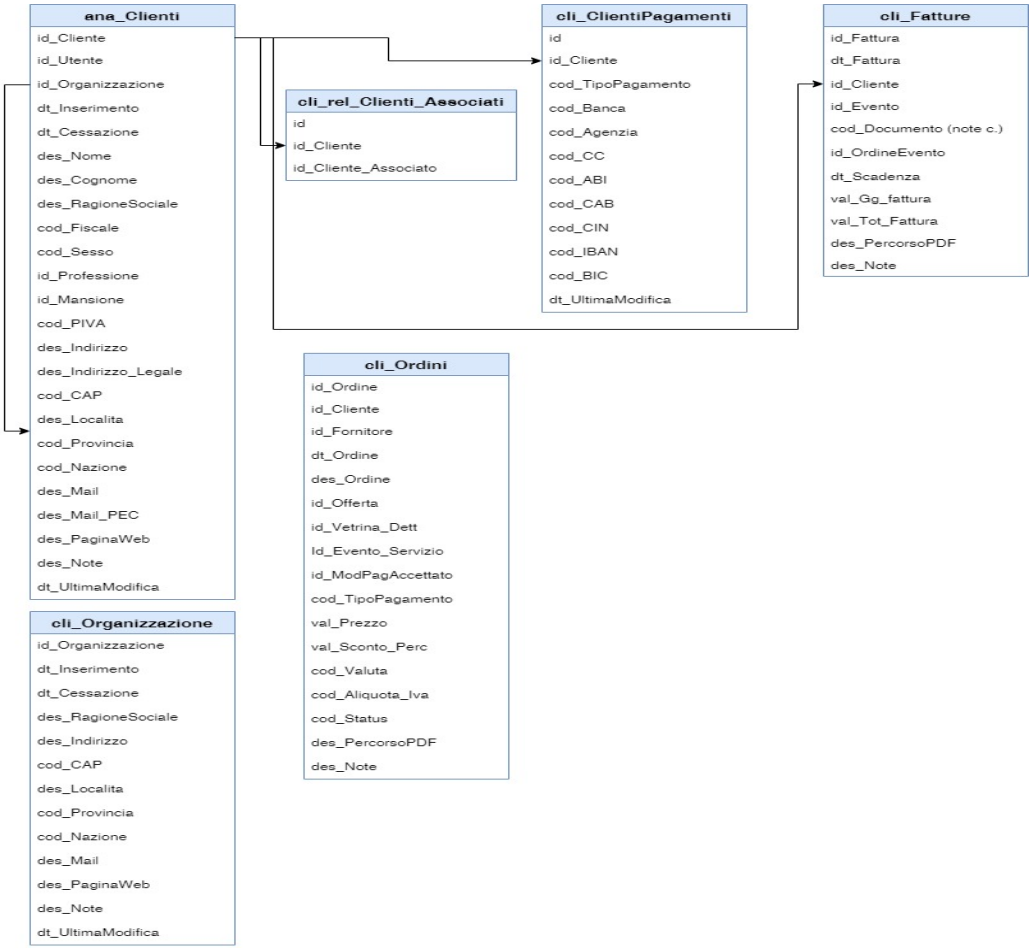


Figura 3.4: Microservizio Cliente

La tabella cli_rel_Clienti_Associati permette l'associazione di uno o più clienti iscritti per la gestione congiunta di un evento, ad esempio marito/moglie/genitori in caso di matrimonio.

- **Microservizio Fornitore:**

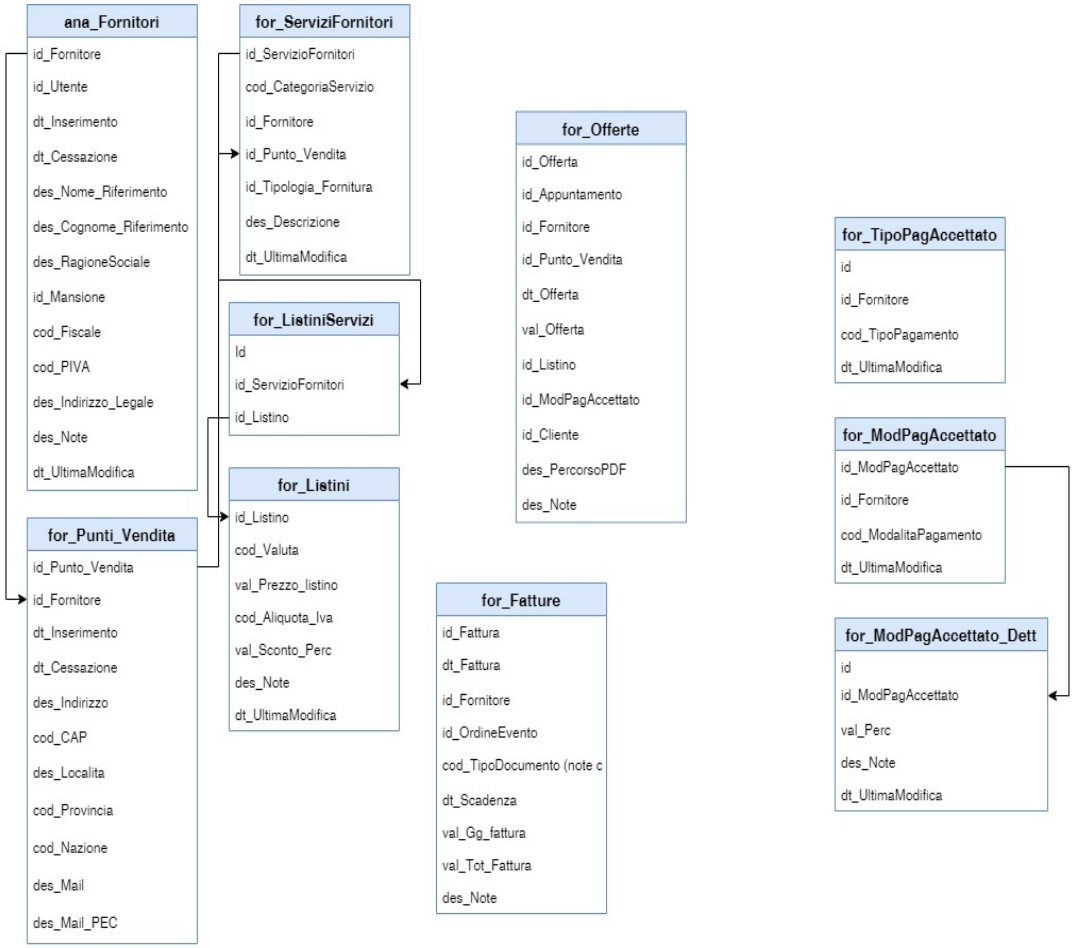
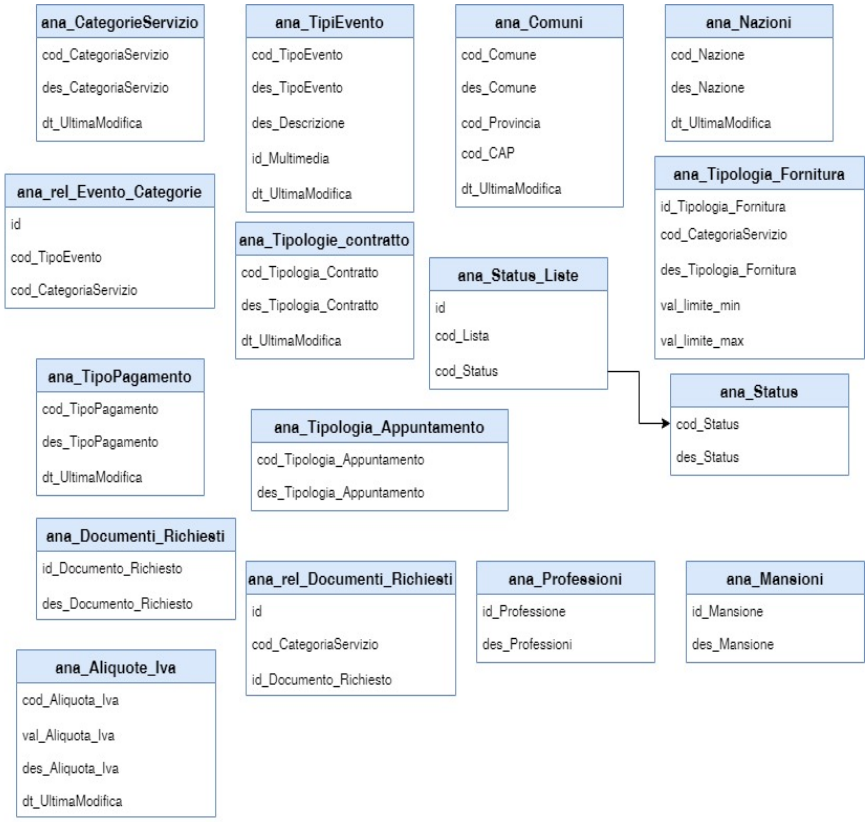


Figura 3.5: Microservizio Fornitore

La tabella for_ServiziFornitori identifica per una Categoria di servizio (Catering, Trasporto, ecc) quali sono i fornitori che ne effettuano il servizio e la tipologia della fornitura (sala fino a 100 persone, ristorante fino a 50 coperti, ecc.

- **Microservizio Anagrafiche:**



Text

Figura 3.6: Microservizio Anagrafiche

La tabella ana_Status_Invito identifica lo status dell'invito (confermato, nessuna risposta, rifiutato, ecc).La tabella ana_Tipologia_Fornitura identifica la peculiarità del servizio : ristorante fino a 30 coperti, sala ricevimento fino a 50 persone, ecc.La tabella ana_rel_Evento_Categorie è una tabella di relazione tra Categorie di servizio e Tipi evento: il servizio di catering può essere offerto sia in un evento Matrimonio sia in una Convention

- **Microservizio Eventi:**

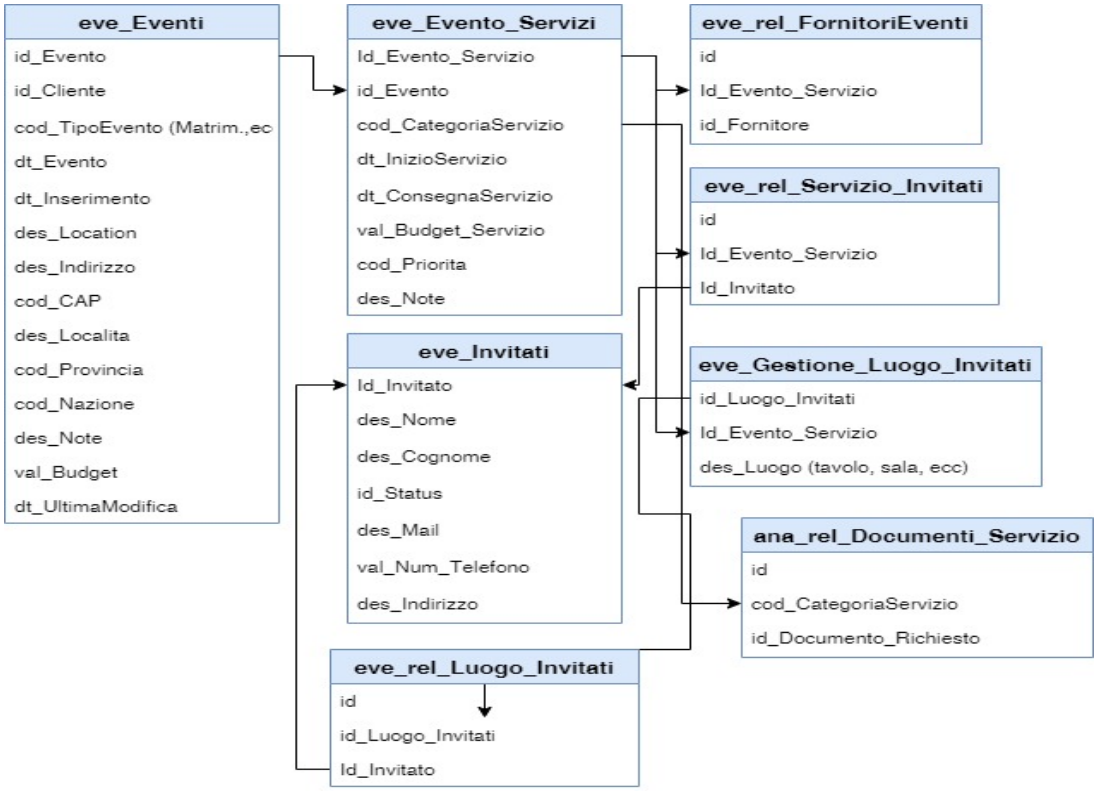


Figura 3.7: Microservizio Eventi

La tabella eve_Evento_Servizi identifica per un evento quali sono i servizi desiderati (cod_CategoriaServizio): Catering, Trasporto, Forniture, ecc. ed il relativo budget. La tabella eve_rel_FornitoriEventi identifica per un servizio prestato (Catering, Trasporto, Forniture, ecc) quali sono i fornitori coinvolti. La tabella eve_Invitati identifica per un servizio prestato(Pranzo,Convention, ecc) gli eventuali invitati. Nel caso specifico potrebbe essere necessario anche gestire la disposizione degli invitati nei tavoli o in sale, e quindi occorre l'ausilio delle tabelle eve_rel_Servizio_Invitati e eve_Gestione_Luogo_Invitati

- **Microservizio Contratti:**

Figura 3.8: Microservizio Contratti

La tabella con_Contratti mantiene tutte le informazioni necessarie per la gestione di una prenotazione. Ogni nuova entry di questa tabella viene creata al momento di una creazione di una nuova prenotazione, inserendo l'id del cliente dell'ordine e del fornitore, aggiungendo i timestamp.
Al variare degli stati del contratto all'interno di questa tabella verranno mantenute le informazioni allineate a quelle della blockchain, in modo tale che il che i servizi possono consumare in maniera più rapida le informazioni utili.

- **Microservizio Vetrine:**

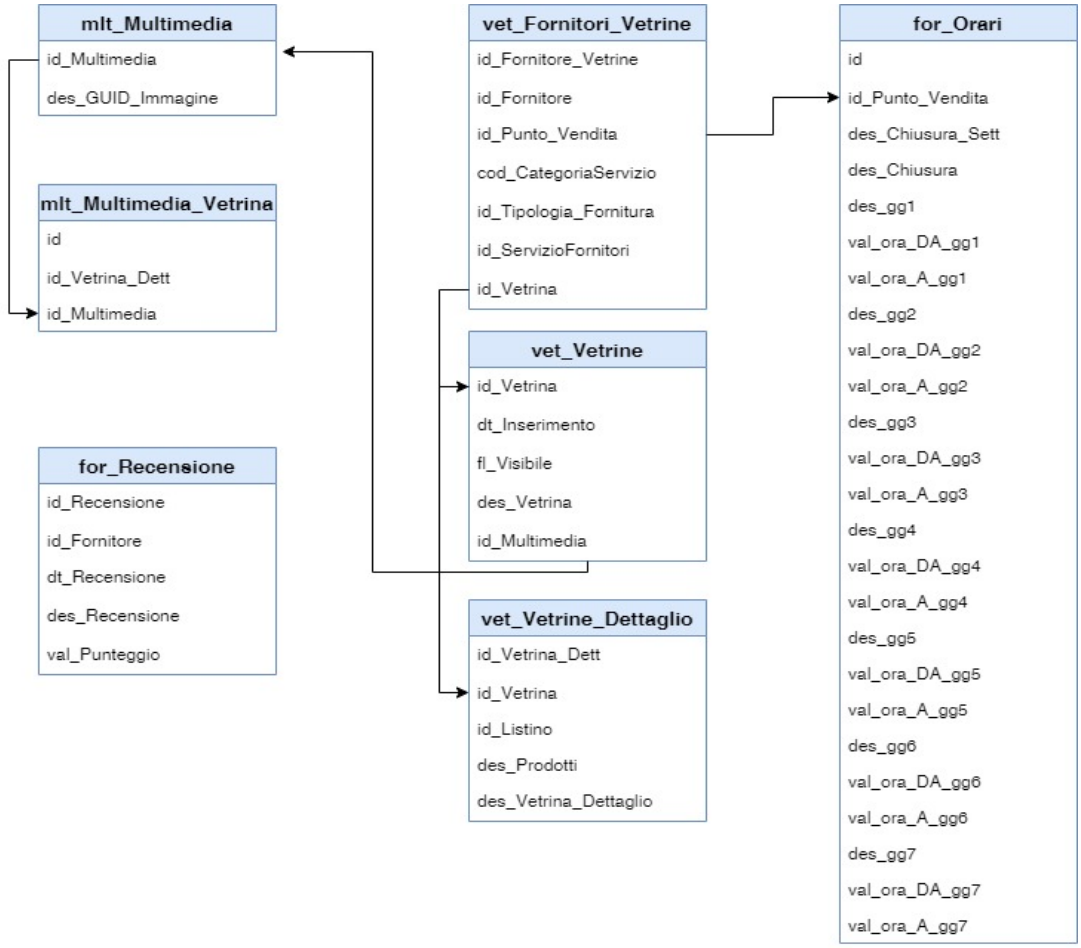


Figura 3.9: Microservizio Vetrine

La tabella vet_Vetrine è la tabella principale del microservizio, tiene traccia delle informazioni principali, al momento della creazione di una nuova vetrina. Quando viene creata una nuova vetrina, si legano vet_Vetrine_Dettaglio che mantiene ulteriori informazioni sul dettaglio della vetrina. Vi si collega entry su vet_Fornitori_Vetrine che detiene le informazioni per legare la vetrina ad un fornitore, un punto vendita, una categoria servizio, ed una tipologia di fornitura. Il punto vendita mantiene le informazioni sugli orari di apertura e chiusura durante la settimana.

- **Microservizio Agenda:**

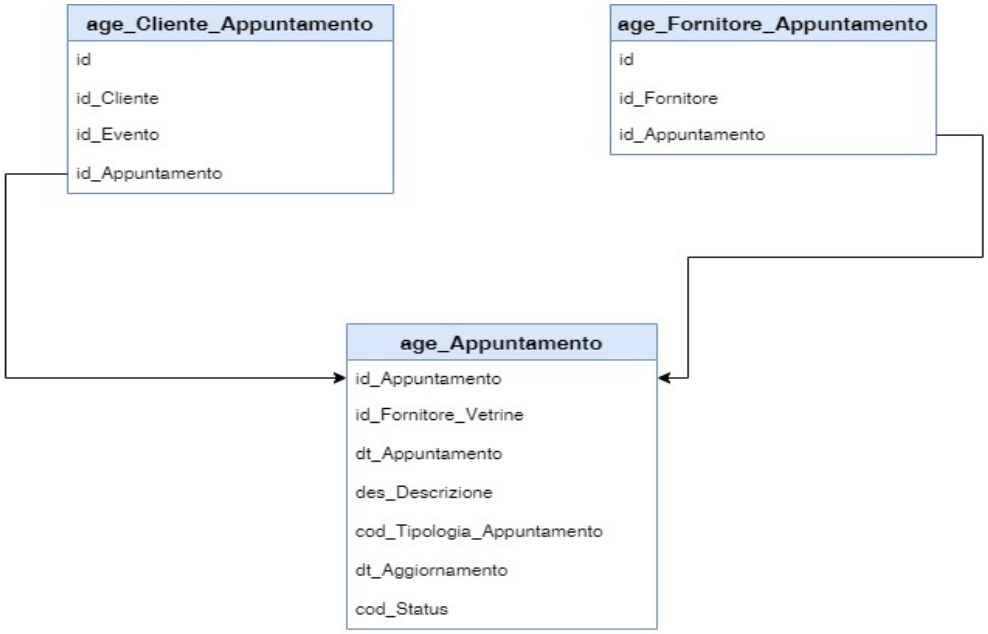


Figura 3.10: Microservizio Agenda

La tabella age_Appuntamento detiene le informazioni all'atto della creazione di un nuovo appuntamento in agenda. Ha informazioni riguardo lo stato dell'appuntamento, ovvero se sono avvenute modifiche nei termini di orari o conferme da parte del fornitore. é collegato tramite id ad una vetrina mediante id_Fornitore_Vetrine ed è possibile aggiungere informazioni sulla descrizione.
Ha anche degli id che gli permettono di tenere traccai del fornitore e del cliente relativi al determinato appuntamento.

3.1.5 Panoramica di distribuzione

L'applicazione web sarà ospitata su un singolo server fisico.

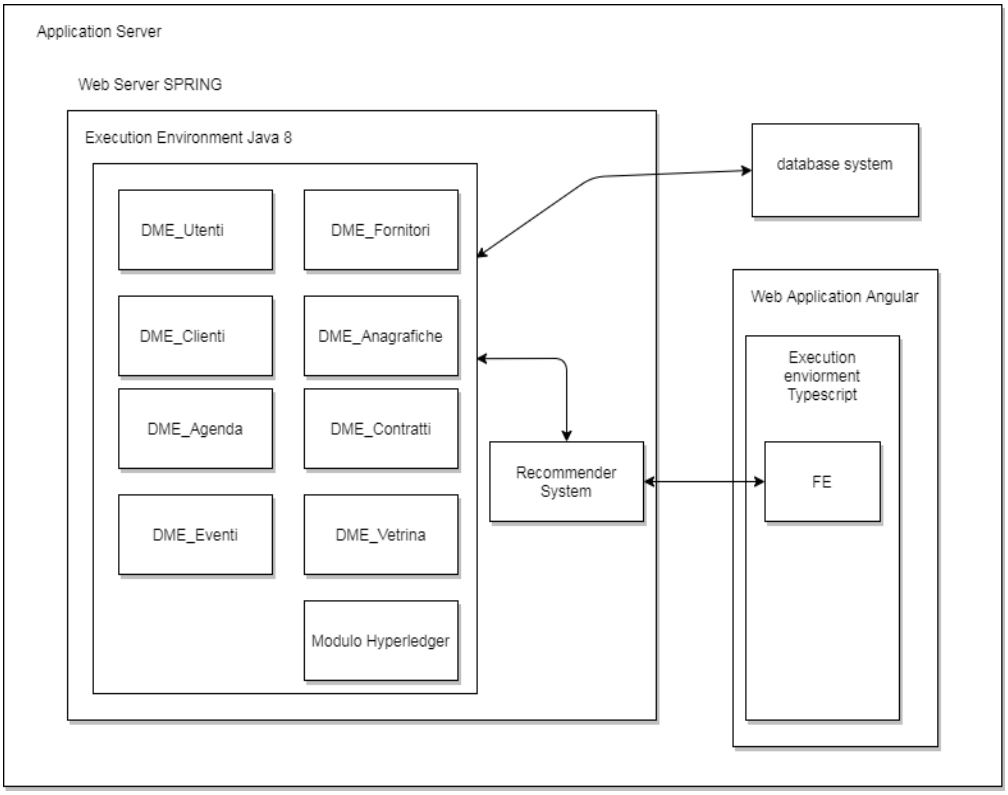


Figura 3.11: Deployment diagram

3.2 Layer Backend

Il lcore della web application nonnchè lo strato infrastrutturale che è realizzato tramite infrastruttura cloud con risorse hardware proprietarie di Linear System.
Oltre alla parte di salvataggio dati e di meccanismo di API per poter alimentare il front end, erano inizialmente previsti due moduli fondamentali per la piattaforma DME:

- Il modulo "Recommender Systems"
- Il modulo "Distributed Ledger / Smart Contract"

Era previsto in fase di progettazione il modulo **recommender system** ma allo stato attuale del progetto non è ancora stato realizzato. Questo avrebbe dovuto occuparsi di avvicinare in modo automatico e puntuale la domanda e l'offerta di servizi relativi agli eventi. In genere i sistemi di raccomandazione trovano applicazione in diversi settori, ed il loro scopo è quello di aiutare le persone ad effettuare scelte basandosi su diversi aspetti.
Data una persona, questi aspetti possono essere ad esempio la propria cronologia, ovvero gli acquisti già effettuati o i voti positivi già dati, o le preferenze di persone simili ad essa. In base a questi aspetti e a come sono prodotte le raccomandazioni i sistemi di raccomandazione si dividono in varie tipologie.
I sistemi di raccomandazione possono essere raggruppati in tre categorie principali, in base all'approccio usato per ottenere le raccomandazioni, più una quarta che è nata in seguito alla crescente diffusione dei social network, e che viene enunciata per ultima:

- Approcci content-based: alla persona saranno raccomandati oggetti simili a quelli che ella ha preferito in passato;
- Approcci collaborativi: alla persona saranno raccomandati oggetti che persone con preferenze simili alle sue hanno preferito in passato;
- Approcci ibridi: questa tipologia racchiude i sistemi di raccomandazione che combinano tecniche usate nelle due precedenti tipologie;
- Approcci semantic-social: si considera un insieme di persone ed uno di oggetti, dove le persone sono connesse da una rete sociale e sia queste ultime sia gli oggetti sono descritti da una tassonomia.

In base alla funzione utilizzata i sistemi di raccomandazione ContentBased e collaborative possono essere:

- Heuristic-based: solitamente prendono come input vari dati e calcolano le raccomandazioni sull'intero dataset delle persone usando metriche di similarità;

- Model-based: costruiscono un modello usando tecniche basate su training set e lo validano su dati di test set. Tale modello viene poi usato per calcolare lo score per gli elementi non ancora raccomandati alle persone. A differenza dei metodi heuristic-based viene dato solo un sottoinsieme di persone attive del dataset come input al modello, che produrr'a da esso le previsioni.

Il modulo **Distributed Ledger/smart contract** si occupa di salvare le transazioni tra clienti e fornitori all'interno di un ledger distribuito e di gestire i contratti in modo digitale dematerializzandoli completamente.
Un ledger distribuito è un registro di transazioni immutabile, gestito all'interno di una rete distribuita di nodi. Ciascun nodo mantiene una copia del registro applicando le transazioni che sono state convalidate da un protocollo di consenso, raggruppate in blocchi, che includono un hash che associa ciascun blocco al blocco precedente.
Per uso aziendale, si considerano generalmente i seguenti requisiti:

- I partecipanti devono essere identificati / identificabili
- Le reti devono essere autorizzate
- Elevate prestazioni di throughput delle transazioni
- Bassa latenza della conferma della transazione
- Privacy e riservatezza delle transazioni e dei dati relativi alle transazioni commerciali

Lo Smart Contract è un contratto scritto in un linguaggio di programmazione di tipo general purpose scritto in modo da determinare, automaticamente, l'esecuzione delle clausole contrattuali all'avverarsi delle determinate condizioni inserite nel contratto stesso.
L'auto esecuzione dello Smart Contract è basata un programma in grado di:

- verificare le clausole che sono state concordate;
- verificare le condizioni concordate;
- eseguire automaticamente le previsioni contrattuali nel momento in cui le informazioni fattuali sono raggiunte e verificate e corrispondono ai dati informatici riferiti alle condizioni e alle clausole previste nel contratto.

3.3 Layer Front-End

Questo strato coinvolge gli aspetti dell'interfaccia verso gli utenti della Web App che implementa DME.

Il layer Front End eroga i servizi di registrazione e accesso sia per quanto riguarda il cliente sia per il fornitore, in classici form di registrazione e accounting. Riconosce in maniera automatica se si tratta di un utente di tipo cliente o fornitore offrendo una pagina customizzata per le differenti tipologie.
Permette l'accesso alle funzionalità previste in fase di analisi come sezioni per la creazione di eventi, per la gestione degli eventi, un'agenda personalizzata legata a tutti gli eventi ma anche al singolo evento, sezioni per gestire il budget e il contratto. Rende operative sia le funzioni di ricerca dei fornitori adeguati al tipo di evento che si deve organizzare che quelle di gestione del contratto direttamente all'interno della piattaforma.
Per quanto riguarda il fornitore, il layer front end rende disponibili oltre le funzionalità di accounting anche quelle per la creazione di un servizio, permettendo il caricamento anche di media associandolo ad una propria vetrina.
La UI è stata progettata mediante l'utilizzo in fase di progettazione e analisi di opportuni mock-up per studiarne la composizione visiva dei vari componenti all'interno dello schermo in cui si visualizza l'interfaccia.
I principi sulla quale si è basata la realizzazione della User Interface sono:

- **Semplicità:** poiché questa piattaforma è indirizzata a qualsiasi tipologia di utente, di qualsiasi età, la vista delle sezioni sono state rese più semplici possibile;

- **Accesso multi-device:** poiché mette degli strumenti a disposizioni utili all'organizzazione giornaliera dell'evento, ci si è concentrato molto sul rendere la Web Application responsive ad ogni tipo di device. Si è posto particolare attenzione sui formati di schermo come il tablet o formati come quelli degli Iphone.

Durante la fase di analisi si è prevista anche un' eventuale realizzazione di un'applicazione mobile in modo da rendere più fruibili i medesimi servizi.
Allo stato attuale non è stata ancora realizzata, tuttavia oltre ovviamente alla realizzazione della struttura dell'applicazione della stessa, quindi ovviamente alla realizzazione di un'interfaccia che si avvicina molto a quella che è la UI della Web Application che è visibile mediante browser da mobile,sarà molto semplice usufruire dei dati poichè i servizi sono resi disponibili come descritto prima da API RESTful.

Capitolo 4

Elenco requisiti e descrizione casi d'uso

4.1 Attori

Nei processi di ingegneria del software, un attore è qualcuno (utente) o qualcosa (hardware o esterno al sistema) che scambia informazioni con il sistema e fornisce i dati di input o riceve dati di output dal sistema. Un attore fa parte parte dell'insieme degli *stakeholders* che sono coinvolti nel software. In particolare, esso specifica un ruolo assunto da un utente o altra entità che interagisce col sistema nell'ambito di un'unità di funzionamento (caso d'uso). é esterno al sistema e può essere un oggetto o una persona.
Nella piattaforma DME gli attori che abbiamo analizzato e riconosciuto si possono individuare con quanto descritto nella tabella sottostante:

Attore	**Descrizione**
Fornitore	Utilizza l'applicazione per inserire i servizi da lui offerti e gestire le richieste di appuntamento e prenotazione
Cliente	Utilizza l'applicazione per organizzare l'evento e prenotare i servizi scegliendo tra quelli proposti
Sistema di pagamento	Viene coinvolto quando si effettua un pagamento all'interno della piattaforma, viene chiamato dal sistema stesso

Tabella 4.1: Attori

Non è da escludere se la piattaforma si dovesse espandere di aggiungere, un attore

che viene individuato come amministratore che potrebbe interagire con il sistema magari modificando o fare operazioni interne.

4.2 Context Diagram e interfacce

4.2.1 Context Diagram

Un context diagram è una vista ad alto livello del sistema che definisce ciò che è all'interno del sistema per essere sviluppato e ciò che è fuori dal sistema come per esempio gli attori.
Nel nostro caso il context diagram di riferimento è sotto riportato:

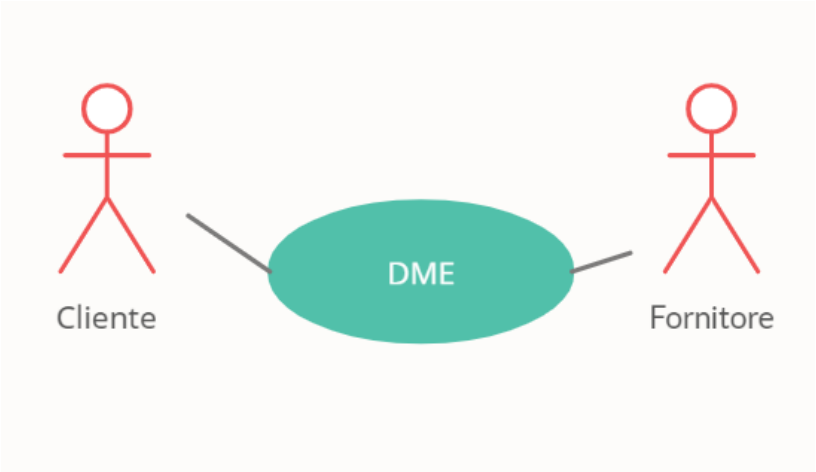


Figura 4.1: Context Diagram

4.2.2 Interfacce

L'interfaccia definisce come avviene l'interazione tra gli attori e il sistema analizzato.
In DME le interfacce analizzate sono:

Attore	Interfaccia logica	Interfaccia fisica
Cliente	GUI	Schermo, tastiera
Fornitore	GUI	Schermo, tastiera

Tabella 4.2: Interfacce

4.3 User story e use cases

Il caso d'uso in informatica è una tecnica usata nei processi di ingegneria del software per effettuare in maniera esaustiva e non ambigua, la raccolta dei requisiti al fine di produrre software di qualità.
Essa consiste nel valutare ogni requisito focalizzandosi sugli attori che interagiscono col sistema, valutandone le varie interazioni. In UML sono rappresentati dagli Use Case Diagram.
Il documento dei casi d'uso individua e descrive gli scenari elementari di utilizzo del sistema da parte degli attori che si interfacciano con esso.

4.3.1 Caso d'uso analizzato

Come detto in precedenza,l'intero progetto, riguarderà la realizzazione di un portale che abbraccerà svariate categorie, dunque in fase di progettazione e di coding si è fatto il più possibile riferimento alla diversa gamma di particolarità di categorie di evento, cercando ove possibile di particolarizzarlo per renderlo il più idoneo possibile alla user experience.
Per fare ciò si è dunque deciso di prendere in considerazione un particolare caso d'uso che poi corrisponde ad un determinato tipo di evento, in modo da poterlo analizzare e studiare ed implementare gli accorgimenti necessari per renderlo caratteristico. Nello specifico si è deciso di prendere in considerazione l'organizzazione di un evento di tipo Matrimonio, in quanto grazie alla particolarità dell'evento e al vasto programma che coinvolge vari ruoli e vari possibili servizi, permette di realizzare un modello "giocattolo" valido per molte altre tipologie di evento .
La piattaforma si è stabilito che fornirà validi strumenti e permette la connessione tra i diversi attori della piattaforma, in maniera real time. Si può quindi organizzare l'evento nella sua interezza, o anche solo una parte di esso, sarà il cliente finale/organizzatore dell'evento a decidere quali fasi organizzare supportati dalla nostra piattaforma. La coppia di sposi avrà a disposizione gli strumenti necessari visualizzare tutta la lista completa degli impegni per organizzare il matrimonio, secondo un piano temporale: attività da svolgere prima, durante e dopo il matrimonio (Agenda), non dimenticando nessun dettaglio dell'evento. Si è pensato che il fornitore potrà ricercare all'interno delle diverse categorie, inviare una richiesta, confrontare le diverse risposte e scegliere quello migliore. Si potrà eseguire tutta la trattativa con il supporto della nostra piattaforma. Inoltre si potrà collegare la lista degli impegni (esempio: prenotazione location), con il budget (costo finale/ costo pagato) ed i fornitori prescelti/prenotati (esempio: Villa Miani), così da tenere sotto controllo anche i costi, e rispettare il budget concordato.

4.3.2 Story

Una volta stabilito lo "use case" è stato redatto un documento in fase di di analisi di requisiti per l'evento di tipo Matrimonio.

"*Filippo e Francesca sono una coppia di futuri sposi che vuole organizzare il proprio matrimonio a Roma nella data del 29 Maggio 2020. Francesca ha individuato online la piattaforma "BestEvents" (da verificare nome) per poter organizzare al meglio l'evento e decide di registrarsi, inserendo i propri dati personali e alcune informazioni sul futuro matrimonio da organizzare. Dopo la registrazione inizia a seguire la lista di impegni che le vengono proposti. Tra le prime attività che Francesca deve definire, insieme a Filippo, c'è l'indicazione del budget di spesa del matrimonio. In questa sessione deve inserire il costo stimato di ogni singola voce di spesa, per ogni fornitore. Alla fine delle attività di selezione e validazione dei diversi fornitori, che avveranno nei mesi a seguire, visualizzerà il costo finale per l'organizzazione di tutto il matrimonio. A seguire decidono di iniziare il percorso di selezione e validazione dei diversi fornitori guidati dall'Agenda personalizzata del loro matrimonio. Tra le prime voci da scegliere c'è la chiesa, e con il portale Francesca riesce a gestire la condivisione dei diversi documenti, atti della cresima, nulla osta, tra le diverse parrocchie. Nei giorni successivi Francesca vuole iniziare a vedere le diverse location nelle quali organizzare il proprio ricevimento e così inizia a visualizzare la lista delle location su Roma, sul portale sono suddivise per categoria: Ville, Agriturismi, Hotel, Ristoranti, Castelli, Spiagge, etc.; selezionando arriva ad una decina di location che le piacciono e vede il dettaglio di ognuna: foto 3D di dettaglio, video, virtual tour, spazi e capienza, servizi offerti, possibilità di alloggio, posizionamento su mappa, eventuale esclusiva di alcuni fornitori, costo di affitto della struttura, n° massimo di invitati, ambienti, recensioni di altre coppie di sposi. Decide di inviare una richiesta di preventivo alle 10 location prescelte, tramite l'app. Ricevute le risposte dalle diverse location valuta la più indicata per il suo matrimonio e conferma la prenotazione per il 29 Maggio al Castello di Tor Crescenza. Il contratto viene archiviato nella sessione dedicata e l'acconto viene pagato sempre tramite app. Successivamente, sempre in base alla scaletta dell'agenda del matrimonio, precedentemente settata, passa alla prenotazione di: catering, allestimenti floreali, fotografo e video, bomboniere, partecipazioni, musica, noleggio auto, animazione adulti e bambini, noleggio arredi e tensostrutture, decorazioni varie, lista di nozze, wedding planner, torte nuziali, abito da sposa, abito da sposo, acconciatura, trucco, fedi. Per ognuno di questi fornitori ha la possibilità di vedere i diversi dettagli e contattare il fornitore per avere maggiori chiarimenti. Si susseguono mesi di ricerche, selezioni ed anche sopralluoghi e prove sul posto. Attraverso la piattaforma Francesca può sempre tenere sotto controllo attività fatte e da fare e aggiornare via via l'agenda con la Finita la prima fase di scelta e prenotazione dei fornitori, contemporaneamente Francesca ha bisogno di gestire gli invitati e quindi li aggiunge*

nella sessione "Invitati" importandoli da Excel, dall'email e dai contatti personali. Li raggruppa per categoria (famiglia, amici, colleghi di lavoro) così sarà più facile aggiungerli ai tavoli. Gli invitati inseriti nella lista appaiono nel "Gestore di Tavoli". Francesca dovrà soltanto disegnare la piantina della sala del ricevimento, selezionarli e trascinarli fino al tavolo assegnato. Potrà verificare l'avanzamento, tramite app, nei giorni successivi di chi ha confermato, chi non ha risposto, e chi non potrà essere presente. Francesca decide di realizzare anche un wedding site personalizzato per inserire informazioni, foto, avanzamenti e idee da condividere con i suoi invitati. Per esempio decide di inserire un sondaggio sul tipo di menù preferito. Qualche giorno prima del matrimonio l'app ricorda a Francesca e a Filippo gli ultimi adempimenti/ appuntamenti da svolgere. Il giorno successivo al matrimonio Francesca e Filippo possono caricare foto e commenti sul loro wedding site. Inoltre possono tenere parenti e amici informati sul loro viaggio di nozze. Francesca e Filippo decidono di lasciare una recensione positiva del servizio offerto da (BestEvents), e inseriscono la lista di tutti i fornitori scelti e caricano le loro foto, così le coppie che dovranno organizzare il matrimonio possono vedere il loro ed eventualmente scegliere gli stessi fornitori (Servizio a favore dei fornitori). "

4.3.3 Use Case Diagram

Figura 4.2: Use case diagram

4.3.4 Use case

Durante la fase di analisi sono stati realizzati diversi use case, che saranno riportati qui di seguito:

1. Creazione account:

Nome use case	Creazione account
ID use case	UC-1
Precondizione/i	Piattaforma "Best Events" disponibile, utente sia in possesso di email valida
Postcondizione	L'Utente ha un account per la piattaforma
Descrizione scenario nominale (sequenza di azioni)	1. L'utente accede alla pagina web e seleziona il form per creare un account; 2. Il sistema mostra la pagina del form; 3. L'utente compila il form con i suoi dati e email corretta; 4. Il sistema verifica il contenuto e che la mail sia valida, in caso positivo manda mail di verifica; 5. L'utente clicca sul link ricevuto per mail e l'account sarà attivato;
Altri scenari non nominali (varianti del nominale)	4.b la mail non è valida, il sistema rimanda al punto 2 mostrando un errore

Tabella 4.3: Use Case Creazione Account

2. Gestione creazione evento privato:

Nome use case	Gestione creazione evento privato
ID use case	UC-2
Precondizione/i	La piattaforma Best Events è disponibile, l'utente è in posseso di un account valido
Postcondizione	L'utente ha selezionato la tipologia del suo evento e adesso lo può gestire
Descrizione scenario nominale (sequenza di azioni)	1. L'utente accede alla piattaforma e clicca il link per effettuare la creazione dell'evento privato; 2. Il sistema mostra gli eventi possibili da creare e il costo relativo al servizio; 3. L'utente seleziona "Matrimonio"; 4. Il sistema mostra le tipologie di pagamento; 5. L'utente ne seleziona una tipologia; 6. Il sistema fa un redirect verso il circuito di pagamento con cui ha fatto l'accordo, in caso positivo notifica l'utente
Altri scenari non nominali (varianti del nominale)	3.a.0 L'utente seleziona "matrimonio" e appare che vuole associare un altro account valido(marito/moglie); 3.a.1 Il sistema verifica che l'account è valido; 6.a.1 Il sistema notifica che c'è stato un errore nel pagamento e viene reindirizzato al punto 4

Tabella 4.4: Use Case Gestione creazione evento privato

3. Gestione attività stima costo:

Nome use case	Gestione attività$_s$*timacosto*
ID use case	UC-3
Precondizione/i	Piattaforma "Best Events" disponibile, utente ha creato l'evento matrimonio
Postcondizione	L'utente ha una stima del costo complessivo
Descrizione scenario nominale (sequenza di azioni)	1. L'utente accede alla pagina web e seleziona la voce: tipologia fornitori e costo stimato; 2. Il sistema mostra la lista della tipologia dei vari fornitori selezionabili e un campo per immettere la stima di quanto si vorrà spendere; 3. L'utente seleziona i vari campi e immette il prezzo stimato; 4. Il sistema mostra la lista delle varie cose da organizzare in ordine di priorità e il costo totale stimato;
Altri scenari non nominali (varianti del nominale)	

Tabella 4.5: Use Case Gestione stima costo

4. Gestione Agenda:

Nome use case	Gestione agenda
ID use case	UC-4
Precondizione/i	Piattaforma "Best Events" disponibile, utente ha settato le tipologie di fornitori da scegliere
Postcondizione	L'utente ha una stima del costo complessivo
Descrizione scenario nominale (sequenza di azioni)	1. L'utente accede alla pagina web e seleziona la voce: agenda; 2. Il sistema mostra la percentuale delle attività completate e in ordine di priorità le attività da svolgere.; 3. L'utente seleziona l'attività da svolgere; 4. Il sistema lo reindirizza alla specifica attività da gestire;
Altri scenari non nominali (varianti del nominale)	

Tabella 4.6: Use Case Gestione agenda

5. Gestione attività prenotazione chiesa:

Nome use case	Gestione attività prenotazione chiesa
ID use case	UC-5
Precondizione/i	Piattaforma "Best Events" disponibile, L'utente ha selezionato dall'agenda l'attività
Postcondizione	L'utente ha prenotato una chiesa
Descrizione scenario nominale (sequenza di azioni)	1. L'utente seleziona il servizio chiesa; 2. Il sistema mostra le chiese disponibili per quella data; 3. L'utente clicca sul tasto ordina per prenotare la chiesa; 4. Il sistema inserisce la prenotazione nel db
Altri scenari non nominali (varianti del nominale)	1.a se l'utente non ha ancora dei documenti pronti salta questa voce

Tabella 4.7: Use Case prenotazione chiesa

6. Gestione attività prenotazione ricevimento:

Nome use case	Gestione attività prenotazione ricevimento
ID use case	UC-6
Precondizione/i	Piattaforma "Best Events" disponibile, L'utente ha selezionato dall'agenda l'attività
Postcondizione	L'utente ha prenotato un ricevimento
Descrizione scenario nominale (sequenza di azioni)	1. Il sistema mostra la lista dei ricevimenti disponibili per quella data e per il numero degli invitati max e min che ha supposto; 2. L'utente visiona la lista e seleziona i luoghi desiderati per ricevere un preventivo; 3. Il sistema appena riceve i preventivi dai ricevimenti notifica l'utente; 4. L'utente prenota e procede al pagamento dell'acconto; 5. Il sistema fa un redirect verso il circuito di pagamento con cui ha fatto l'accordo, in caso positivo notifica l'utente, dopodichè genera il contratto e lo inserisce in "Contratti";
Altri scenari non nominali (varianti del nominale)	2.a L'utente può selezionare il luogo e il sistema lo reindirizzerà in una pagina della piattaforma che mostrerà tutte le caratteristiche; 6.a Se il sistema riceve richiesta di sopralluogo notifica il ricevimento per accordarsi sulla data;

Tabella 4.8: Use case relativo alla prenotazione del ricevimento

7. Gestione contratti:

Nome use case	Gestione contratti
ID use case	UC-7
Precondizione/i	Piattaforma "Best Events" disponibile
Postcondizione	L'utente ha visualizzato i contratti
Descrizione scenario nominale (sequenza di azioni)	1. L'utente selezione la voce contratti; 2. Il sistema mostra la lista dei contratti stipulati e se sono conclusi o in sospeso; 3. L'utente può selezionare un contratto per scaricarlo o avere ulteriori dettagli;
Altri scenari non nominali (varianti del nominale)	

Tabella 4.9: Use case relativo alla gestione dei contratti

8. Gestione invitati:

Nome use case	Gestione invitati
ID use case	UC-8
Precondizione/i	Piattaforma "Best Events" disponibile
Postcondizione	L'utente ha gestito gli invitati
Descrizione scenario nominale (sequenza di azioni)	1. L'utente seleziona la voce invitati; 2. Il sistema mostra la lista degli invitati ; 3. L'utente può selezionare un invitato e selezionare il suo stato(confermato, rifiuto); 4. L'utente può inserire una piantina per la gestione dei tavoli ; 5. Il sistema visualizza la piantina dei tavoli e permette l'inserimento degli invitati;
Altri scenari non nominali (varianti del nominale)	2.a Se l'utente ancora non aveva caricato la lista, il sistema mostra le varie possibilità di inserimento e l'ordinamento secondo gruppi

Tabella 4.10: Use case relativo alla gestione degli invitati

4.4 Requisiti funzionali

Un requisito funzionale, nell'ambito dell'ingegneria del software, è un requisito che definisce una funzione di un sistema di uno o più dei suoi componenti, definendone la tipologia degli ingressi e delle uscite, nonché il comportamento.
Per ogni requisito abbiamo identificato una priorità secondo il seguente schema:

- Critico (C): Il requisito deve essere soddisfatto per il corretto funzionamento del sistema
- Standard (S): Il requisito dovrebbe essere soddisfatto ma la sua assenza non impatta sul funzionamento del sistema
- Opzionale (O): Il requisito può essere soddisfatto ma la sua assenza non impatta sul funzionamento del sistema

Nella piattaforma DME i requisiti funzionali che abbiamo analizzato risultano:

ID	Descrizione	Priorità
FR1	Gestione tipologia utente	C
FR2	Registrazione e memorizzazione dati cliente	S
FR3	Registrazione e memorizzazione dati fornitore	S
FR4	Memorizzazione di un servizio, la sua tipologia e la relativa vetrina da parte di un fornitore	C
FR5	Creazione e memorizzazione evento da parte di un cliente	C
FR6	Memorizzazione budget da parte del cliente	O
FR7	Visualizzazione e gestione dell'agenda da parte del cliente e del fornitore	S
FR8	Memorizzazione ordine del servizio scelto dal cliente	S
FR9	Inserimento e memorizzazione nominativi invitati dal cliente	S
FR10	Gestione e assegnazione tavoli agli invitati del cliente	S
FR11	Memorizzazione e gestione dello smart contract	C
FR12	Memorizzazione e gestione pagamento servizio	C

Tabella 4.11: Requisiti funzionali

4.5 Requisiti non funzionali

I requisiti non funzionali rappresentano i vincoli e le caratteristiche relative al sistema, come vincoli di natura temporale, vincoli sul processo di sviluppo e sugli

standard da adottare. I requisiti non funzionali non riguardano solo il sistema software che si sta sviluppando, alcuni possono vincolare il processo usato per sviluppare il sistema. Essi non riguardano direttamente le specifiche funzioni fornite dal sistema, ma possono sia riferirsi a caratteristiche che si desidera il sistema presenti sia definire i vincoli ai quali il sistema deve sottostare.

Nella piattaforma DME i requisiti non funzionali che abbiamo analizzato risultano:

ID	**Tipo**	**Descrizione**	**Priorità**
NFR1	Portabilità	l'applicazione può essere eseguita su qualunque browser	S
NFR2	Portabilità	l'applicazione può essere eseguita su qualunque dispositivo	S

Tabella 4.12: Requisiti non funzionali

4.5.1 Mapping tra requisiti e casi d'uso

	UC-1	**UC-2**	**UC-3**	**UC-4**	**UC-5**	**UC-6**	**UC-7**	**UC-8**
FR1	X							
FR2	X							
FR3	X							
FR4					X	X		
FR5		X						
FR6		X	X					
FR7				X				
FR8					X	X	X	
FR9								X
FR10								X
FR11							X	
FR12							X	

Tabella 4.13: Mapping tra requisiti e casi d'uso

Capitolo 5

Distributed Ledger e smart contract

5.1 DLT: La distributed ledger technology

Si tratta dei sistemi cosiddetti "distribuiti", che sono caratterizzati dalla replica sincronizzata del registro in diversi punti detti nodi, che si comportano come sistemi indipendenti collegati tra loro da una rete di comunicazione, in cui ciascun nodo gestisce la propria copia, mantenuta allineata con le altre.
Ciò che contraddistingue la DLT da una replica di informazioni accessibili localmente dal singolo nodo in lettura ma modificate attraverso l'intervento di un ente centrale che determina quali sono gli aggiornamenti da distribuire a tutti i nodi della rete , è la capacità da parte dei sistemi che la adottano di gestire anche le modifiche al registro stesso in maniera distribuita.Ogni aggiornamento del registro è regolato tramite meccanismi di consenso che permettono alla rete di convergere su un'unica versione del registro stesso. Nonostante che i soggetti all'interno della rete modificano in maniera indipendente le informazioni.
Le caratteristiche più evidenti che permettono di distinguere i diversi sistemi di distributed ledger coinvolgono tre aspetti principali:

- La tipologia di rete che ne determina quali attori ne fanno parte e il ruolo nel processo di validazione;

- Il meccanismo di consenso, che determina le modalità con cui la rete aggiorna il registro;

- La struttura del registro che definisce come le informazioni vengono organizzate all'interno del registro;

In base alla tipologia della rete che è stata scelta ci si può distinguere tra diverse categorie di sistemi:

- *public* ai quali chiunque può accedere per leggere i dati delle transazioni senza bisogno di un'esplicita approvazione da parte di un soggetto;
- *private* in cui, al contrario, esiste un ente centrale che gestisce l'accesso ai dati , limitando solo a a un insieme di utenti autorizzati.

In base al ruolo che viene attribuito agli attori nel processo di validazione si distinguono sistemi:

- *permissionless*: tutti gli attori possono accedere ai dati, validano le transazioni, senza particolari autorizzazioni;
- *permissioned:* solo determinati attori sono autorizzati a svolgere le operazioni

Il sistema che è stato realizzato, si può dire che si è scelta una categoria di sistema di tipo privata con permessi.
Infine per completezza vi è una terza caratteristica dei sistemi distributed ledger che è la struttura del registro, ovvero come le informazioni sono organizzate all'interno del registro:

- *ledger tradizionale* in cui le informazioni sono validate e registrate a livello di singola transazione;
- *catena di blocchi*, modalità in cui le transazioni vengono raccolte all'interno di un blocco che viene validato e registrato a livello aggregato con un altro;

5.2 Blockchain

La tecnologia blockchain rappresenta una particolare scelta implementativa di registro distribuito che è inclusa nella famiglia delle tecnologie di distributed ledger. La particolare struttura del registro permette però di aggiungere alcune funzionalità che altre soluzioni, non sono in grado di abilitare.
La blockchain è un registro condiviso e immutabile che facilita il processo di registrazione delle transazioni e di tracciamento degli asset in una rete di business. Un asset può essere tangibile (case, automobili, soldi, terre) o intangibile (proprietà intellettuali, brevetti, copyright, marchi). Praticamente qualsiasi cosa che abbia un valore può essere rintracciata e scambiata su una rete blockchain, riducendo i rischi e i costi per tutti gli interessati.
Il business dipende dalle informazioni. Più sono rapide e accurate, meglio è. La blockchain è l'ideale per distribuire le informazioni in quanto fornisce informazioni immediate, condivise e completamente trasparenti, memorizzate su un registro

immutabile a cui possono accedere solamente i membri di rete con autorizzazione. Una rete blockchain può tenere traccia di ordini, pagamenti, conti, produzione e molto altro ancora. E poiché i membri condividono una visione singola della verità, è possibile vedere tutti i dettagli di una transazione end-to-end, generando così maggiore fiducia, oltre a nuove efficienze e opportunità.
[3] Nel caso di sistemi blockchain si aggiungono due ulteriori elementi che permettono di distinguere le diverse tipologie: i trasferimenti, che determinano le caratteristiche delle transazioni gestite attraverso il registro distribuito e gli asset, che determinano l'oggetto del trasferimento all'interno del sistema.

Elementi chiave di una blockchain

- **Immutabilità dei record:**I partecipanti alla blockchain non potranno modificare nessuna transazione che è stata registrata nel registro condiviso, nessun malintenzionato potrà manomettere le transizioni. Per poter gestire un errore o per modificare una transazione o solo u record di essa, sarà necessaria registrare una nuova transazione.
  Il meccanismo per cui il registro è immutabile è che essendo una catena di blocchi, a ciascun blocco viene inserita un' "impronta" digitale crittografica del blocco precedente. Segue che per modificare una singola informazione è necessario modificare anche il resto della catena.

- **Registro distribuito:**I partecipanti alla rete hanno accesso al registro distribuito e al record immutabile di transazioni in esso contenuto. Con questo registro condiviso, le transazioni vengono registrate una sola volta, eliminando la duplicazione dei compiti, tipica delle reti di business tradizionali.

- **Contratti smart:** Il contratto "intelligente" è memorizzato come una serie di regole che verranno eseguite automaticamente. Questo può definire le condizioni per i trasferimenti di obbligazioni aziendali, includere le condizioni per l'assicurazione di viaggio da pagare e molto altro.

Vantaggi Le caratteristiche intrinseche di una tecnologia blockchain, in particolare le peculiarità che riguardano la sicurezza nelle operazioni che si possono svolgere all'interno di essa ha portato a molte aziende del mercato ad investire su questa tecnologia. Tra queste l'IBM che riporta: *le operazioni spesso sprecano energie nella conservazione di record duplicati e convalide di terze parti. I sistemi di conservazione dei record possono essere vulnerabili a frodi e attacchi informatici. La trasparenza limitata può rallentare la verifica dei dati. E con l'arrivo dell'IoT, i volumi delle transazioni sono esplosi. Tutto questo rallenta il business e riduce il fatturato – e ci fa capire che dobbiamo adottare nuovi metodi.*
Maggiore fiducia: Con la blockchain, in qualità di membro di una rete di soli

membri, puoi stare tranquillo di ricevere dati accurati e tempestivi, e di condividere i record personali della blockchain solamente con i membri di rete a cui avete specificamente concesso l'accesso.
Maggiore sicurezza:Il consenso sull'accuratezza dei dati è richiesto a tutti i membri della rete e tutte le transazioni convalidate sono immutabili: registrate in modo permanente. Nessuno, nemmeno un amministratore di sistema, può eliminare una transazione.
Più efficienze: Con un registro distribuito che viene condiviso tra i membri di una rete, vengono eliminate le inutili riconciliazioni di record. E per velocizzare le transazioni, è possibile memorizzare sulla blockchain una serie di regole – chiamata contratto intelligente – ad esecuzione automatica.
[3] Questo ci fa capire l'importanza attuale di una blockchain, per la quale colossi come l'IBM stano molto investendo in termini di risorse finanziarie, in quanto ne riconoscono i vantaggi.

5.3 Smart Contract

5.3.1 Cenni storici e definizione

Il primo a parlare di smart contract è stato Nick Szabo nel 1994. Egli definisce lo smart contract come *Uno smart contract è un insieme di impegni, specificati in forma digitale, inclusi i protocolli all'interno dei quali le parti mantengono questi impegni. I protocolli sono solitamente implementati con programmi su una rete di computer, o in altre forme di elettronica digitale, quindi questi contratti sono "più intelligenti" dei loro antenati cartacei ".*
Gli Smart Contract vengono definiti come clausole contrattuali incorporate nell-l'hardware e nel software in modo tale da rendere quasi impossibile la violazione.

5.3.2 Caratteristiche di funzionamento

Fondamentale nel funzionamento degli smart contract è il meccanismo *if-then-else* che una determinata condizione del codice degli smart contract venga eseguita solo al verificarsi di un determinato evento.
Come dice anche Szabo durante l'esecuzione dello smart contract non viene utilizzata nessuna intelligenza artificiale. Infatti il codice dello smart contract viene eseguito solo quando viene attivato da una transazione inviata da una persona fisica o da un messaggio inoltrato da un altro smart contract.
Lo smart contract è una tecnologia che utilizzando la DLT e in particolar modo la blockchain permette di realizzare un processo negoziale capace di eseguirsi in maniera autonoma.

5.3.3 Normativa italiana sugli smart contract

Dal punto di vista giuridico lo Smart Contract è una questione aperta, seppur tratta in Italia.
Una delle questioni principali è che uno Smart Contract per come è stato pensato non dispone di nessuna capacità di "autodeterminarsi" o di adattarsi ai mutamenti della realtà, ovvero non è intelligente in senso proprio. Con un esempio, possiamo pensare al caso della prestazione contrattuale che, in un contratto ad esecuzione differita, diventa eccessivamente onerosa ai sensi dell'art 1467 c.c.. In questo caso lo smart contract non ha la funzionalità di sospendere l'esecuzione in maniera autonoma a meno che non vi sia una disattivazione dello script per mezzo di una specifica istruzione di un attore.
Essendo in evoluzione e comunque una tecnologia in fase esperimentale non si esclude che in futuro non ci sia la possibilità che lo smart contract sia dotato di intelligenza artificiale o di una sorta di capacità adattiva in modo tale che possa assumere delle decisioni non previste in fase di programmazione autonomamente e magari rinegoziare i termini contrattuali.
Dunque da un punto di vista legale allora come si pone questa tecnologia? A.Cerrato in Smart Contract e blockchain ne descrive la questione in maniera puntuale: *"Il contributo intende prima di tutto inquadrare da un punto di vista giuridico la figura dello « smart contract » anche in relazione al contesto storico e normativo attuale. Emergerà che lo smart contract è uno strumento tecnologicamente avanzato e versatile a disposizione delle pani che possono servirsene, nel contesto di un rapporto contrattuale, per finalità diverse: come mero veicolo di scambio delle dichiarazioni negoziali, come si farebbe con una e-mail certificata ; come mezzo di attuazione del contralto concluso in forma « tradizionale »; ovvero come fonte esso stesso del vincolo negoziale, rendendo quindi lo smart contract il « contratto » . Quest'ultima possibilità apre scenari finora inesplorati per il diritto dei contratti, chiamato alla « prova di resistenza » di fronte alla sfida (culturale, prima di tutto) di una tecnologia che promette di disintermediare i rapporti commerciali, azzerare il rischio di inadempimento, liberare le parti dalla schiavitù della fiducia reciproca grazie (sembra quasi un paradosso...) alle « catene di blocchi » che formano un registro distribuito fra innumerevoli nodi indipendenti che certificano le transazioni rendendole immutabili. In che misura le norme vigenti (anche di proprietà intellettuale) siano adeguate a governare questo fenomeno è un interrogativo al quale, in questo stadio ancora embrionale di diffusione nella prassi dei commerci, non è agevole dare risposta.*

Aspetti concettuali e culturali Perchè non si può considerare un contratto tradizionale?

- L'uso dello smart contract non ammette equipollenti: il linguaggio informatico con tutto ciò che ne discende intrinsecamente, è impossibilitato a tradurre le istruzioni eseguibili nelle consuete clausole generali del diritto (buona fede, correttezza, deligenza);

- Il secondo elemento, che riguarda un aspetto "tipologico" dello smart contract, cioè con l'utilizzo della blockchain vi è la conseguente disintermediazione totale dei rapporti negoziali tra le parti e la mancanza del'elemento fiduciario. In quanto la blockchain neutralizza l'interferenza dei comportamenti umani e ne limita l'inadempimento delle clausole. Si può dire che le transazioni su blockchain non sono disintermediate ma diversamente intermediate. Anche se tuttavia questo potrebbe essere visto solo come un gran vantaggio;

- Il terzo elemento, seppur potrebbe sembrare in netto parallelismo con un contratto tradizionale non è proprio del tutto così. Di primo acchito potrebbe sembrare che l'automaticità del codice di uno smart contract e in particolare l'algoritmo *"if..then..else* potrebbe richiamare la figura della condizione o del termine, cioè la clausola accessoria che fa dipendere il dispiegarsi degli effetti del contratto ovvero la sua risoluzione al verificarsi di un evento o al sopraggiungere di una data. L'ineluttabilità dell'esecuzione dello smart contract fa perdere il suo appeal, fin quando almeno non saranno del tutto "intelligenti", perchè troppo rigidi rispetto ad un contratto tradizionale. Nel senso che, seppur è pur che elimina qualsiasi possibilità di inadempimento da parte del creditore, ma priva il debitore del diritto in via giudiziaria di rimodulazione equitativa della prestazione dovuta.

In sostanza il concetto per cui questa tecnologia non è stata del tutto digerita e considerata come un contratto tradizionale, è la sua rigidità intrinseca nella certezza di esecuzione.

Norma italiana Con l'entrata in vigore del Decreto Semplificazioni nel Febbraio del 2019, l'Italia è diventata il secondo Paese dell'Unione Europea dopo Malta e comunque uno dei primi Paesi al Mondo a riconoscere giuridicamente uno smart contract.
L'articolo *8-ter*, comma secondo del Decreto Semplificazioni vi si riporta dunque definito lo smart contract: *Si definisce smart contract un programma per elaboratore che opera su tecnologie basate su registri distribuiti e la cui esecuzione vincola automaticamente due o più parti sulla base di effetti predefiniti dalle stesse".*
Dunque la norma italiana non prevede una definizione tecnica dal punto di vista informatico di ciò che deve essere uno Smart Contract per essere definito tale. Tuttavia da quello che si delinea dalla norma è il contratto intelligente non è un software qualunque ma deve essere eseguito necessariamente su DLT, definizione

che si lega strettamente con la definizione fatta da altri legislatori, che si sono occupati della materia, su il *Whitepaper Ethereum.* L'altro vincolo che risalta nella norma è la precisazione che il contratto vincola automaticamente due o più parti.
La seconda parte dell'emendamento afferma che: *Gli smart contract soddisfano il requisito della forma scritta previa identificazione informatica delle parti interessate, attraverso un processo avente i requisiti fissati dall'Agenzia per l'Italia digitale con linee guida da adottare entro novanta giorni dalla data di entrata in vigore della legge di conversione del presente decreto* .
Ovvero significa che sono fissate le condizioni in presenza delle quali uno smart contract, soddisfa la forma scritta. La forma scritta ha una rilevanza importante nel nostro ordinamento in quanto può essere richiesta ai fini di validità (ad substaniiam) o di prova (ad probalionem) di un contratto. Poichè uno smart contract viene considerato come un documento informatico elettronico, l'art.20 comme 1-bis CAD indica l'efficacia probatoria del documento informatico, di conseguenza la validità di uno smart contract a livello giuridico. Questione che non discorda con quanto detto in precedenza riguardo il parallelismo con un contratto tradizionale, in quanto erano considerazioni di tipo più "formale".
Per concludere e per precisare secondo quanto dice la legge italiana il requisito fondamentale è la forma, cioè qualsiasi esternazione che fa "conoscere al mondo" che c'è questo accordo, però la forma scritta è requisito necessario quando è prescritta dalla legge, che prevede la scrittura per alcuni contratti, ad esempio è prevista per il contratto di compravendita di un immobile, diversamente il contratto è nullo, ed è richiesta la forma scritta sotto forma di atto pubblico (che è redatto da un pubblico ufficiale o un notaio) o una scrittura privata (per intenderci un contratto scritto dal venditore e compratore corredato dei dati, documenti e firma).

5.4 Smart Contract in DME

L'innovazione di Digital Management of Event non riguarda solamente la digitalizzazione di un impianto che permetta di avere svariate funzionalità a livello organizzativo, ma riguarda anche la possibilità per l'utente di gestire l'intero processo: ovvero di selezionare e contrattualizzare la location, di contattare e acquistare un servizio dai vari fornitori che saranno iscritti alla piattaforma. Tutto questo sarà quindi digitalizzato e di conseguenza i possibili vincoli relativi alla privacy e alla contrattualistica digitale saranno implementati con gli Smart Contract. La chiusura dei contratti direttamente all'interno della piattaforma attraverso "smart contract" è destinata, come dimostrato ampiamente dalla analisi di mercato, ad incrementarsi fortemente in relazione alle esigenze delle aziende produttrici di implementare soluzioni per la trasformazione digitale delle imprese, da un lato, e per la ubiquità multidevice ed in mobilità, dei servizi da erogare.

L'utilizzo degli Smart Contract è coadiuviato dall'utilizzo di un distributed ledger per permettere la comunicazione dei diversi attori della supply chain.

Il modulo "Distributed Ledger/smart contract" si occupa di salvare le transazioni tra clienti e fornitori all'interno di un ledger distribuito e di gestire i contratti in modo digitale dematerializzandoli completamente. Il ledger distribuito mantiene un registro di transazioni immutabile, gestito all'interno di una rete distribuita di nodi. Ciascun nodo mantiene una copia del registro applicando le transazioni che sono state convalidate da un protocollo di consenso, raggruppate in blocchi, che includono un hash che associa ciascun blocco al blocco precedente.

Per rendere quindi l'esperienza completa e innovativa l'utilizzo di questa tecnologia è fondamentale. Tuttavia nonostante esista e sia utilizzato in alcuni rami ancora non c'è un utilizzo ampissimo di questo ecosistema basato su i ledger e smart contract dunque si può considerare comunque un'implementazione in fase sperimentale che sicuramente con il passare degli anni sarà sempre più utilizzato. Attualmente infatti, non ci sono documentazioni accademiche a riguardo quindi tutto ciò che si è implementato si è basato su documentazioni ufficiali delle varie tecnologie utilizzate e sullo studio e analisi di ciò che poteva essere più adatto come soluzioni per la nostra piattaforma DME.

Considerazioni giuridiche in DME Riprendendo le considerazioni di A.Cerrato su riportate, si può dire che per quanto riguarda DME le scelte implementative che toccano aspetti giuridici si avvicinano alla prima delle posizioni riportate.
Lo smart contract viene visto come uno strumento negoziale certificato , a cui vengono posti dei vincoli e che le parti (clienti e fornitori) possono farne assoluto affidamento in quanto le due parti. In particolare il fornitore azzera il rischio di inadempimento da parte del cliente, in quanto l'effettiva chiusura del contratto avviene solo dopo aver effettuato tutti i pagamenti ricevuti.
Per adesso il nostro smart contract si può essenzialmente definire come uno strumenti di esercizio di attività contrattuale però non è escluso, che la logica dello smart contract possa essere integrata magari ponendo altri vincoli o altre assicurazioni, ma avvicinarsi a tutto quegli aspetti che riguardano un contratto tradizionale è una sfida e un processo che non sarà solo di DME.

5.5 Stato dell'arte e strumenti utilizzati

Nella seguente sezione si darà una panoramica degli strumenti esistenti utilizzati per lo sviluppo della web application.

5.5.1 Hyperledger

Nato da un progetto della fondazione Linux nel 2015, Hyperledger è una tecnologia open source implementata tramite Blockchain. Lo scopo di Hyperledger è quello di implementare attraverso i suoi framework e strumenti, un Business Network Model. Una blockchain è un registro (ledger) di transazioni immutabile, gestito all'interno di una rete distribuita di nodi. Ciascun nodo mantiene una copia del ledger applicando le transazioni che sono state convalidate da un protocollo di consenso, raggruppate in blocchi che includono un hash che associa ciascun blocco al blocco precedente. Per l'utilizzo che dobbiamo farne, poichè poi sarà rivolto ad un utilizzo commerciale, dobbiamo considerare i seguenti requisiti:

- I partecipanti devono essere identificati / identificabili
- Le reti devono essere autorizzate
- Elevate prestazioni di throughput delle transazioni
- Bassa latenza della conferma della transazione
- Privacy e riservatezza delle transazioni e dei dati relativi alle transazioni commerciali

Fabric

Uno dei framework dell'ecosistema Hyperledger è Fabric, progettata per l'utilizzo in contesti aziendali, che offre alcune funzionalità di differenziazione chiave rispetto alle altre piattaforme di blockchain più diffuse.

Fabric ha un design modulare e versatile soddisfa un'ampia gamma di casi d'uso del settore, [4] tra cui banche, finanza, assicurazioni, sanità, risorse umane e supply chain e quindi molto adatto anche per il nostro caso studio. Fabric è la prima distribuited ledger platform a supportare smart contracts creati in linguaggi di programmazione generici come Java, Go e Node.js, piut- tosto che linguaggi specifici del dominio vincolati (DSL). Ciò significa che la maggior parte delle imprese dispone già delle competenze necessarie per svilup- pare contratti intelligenti e non è necessaria alcuna formazione aggiuntiva per apprendere una nuova lingua o DSL. Anche la piattaforma Fabric è autorizzata, nel senso che a differenza di una rete pubblica senza autorizzazione, i partecipanti nella rete sono noti l'uno all'altro. Ciò significa che, sebbene i partecipanti non possano fidarsi pienamente l'un l'altro (possono, ad esempio, essere concorrenti nello stesso settore), una rete può essere gestita secondo un modello di governance che si basa sul fatto che la fiducia esiste tra i partecipanti, come, ad esempio, un accordo per la gestione delle controversie. Uno dei più importanti differenziatori della piattaforma è il supporto per protocolli di consenso collegabili che consentono alla piattaforma di essere

più efficacemente personalizzata per adattarsi a casi d'uso particolari e modelli fiducia, come potrebbe essere un contratto. Fabric utilizza protocolli di consenso per incoraggiare l'esecuzione di smart contracts. Il modello Hyperledger Fabric è composto da vari elemnti.

Asset Rappresentano gli oggetti al centro dello scambio fra due entità. Consentono lo scambio di quasi tutto ciò che possiede un valore monetario attraverso la rete. Gli Asset possono essere tangibili (oggetti reali) o intangibili (contratti e proprietà intellettuale). Hyperledger Fabric permette di modificare gli asset utilizzando le transazioni. Sono rappresentati come una collezione di coppie chiave-valore, ed i loro cambi di stato sono registrati come transazioni in un Channel ledger. Possono essere rappresentati in binario e/o in formato JSON.

Chaincode Chaincode è un software che definisce gli asset e le istruzioni delle transazioni per modificare tali asset; dunque, rappresenta la logica di business. Gestisce le regole per leggere o modificare le coppie chiave-valore o altre informazioni di stato del database. Le funzioni di Chaincode vegono eseguite sullo stato corrente del ledger e vengono inizializzate tramnite una proposta di transazione. Il risultato dell'esecuzione di tali funzioni è un set di scritture chiave-valore che può essere inviato alla rete e applicato al registro su tutti i nodi.

Ledger Il ledger è il record ordinato di tutte le transizioni di stato nel sistema. Le tran- sizioni di stato sono il risultato di invocazioni chaincode ("transazioni") presentate dalle parti partecipanti. Ogni transazione genera un insieme di asset, come coppie chiave-valore, che sono inserite nel ledger come creazioni, aggiornamenti o eliminazioni. Il registro è costituito da una blockchain ("catena") per memorizzare il record immutabile e sequenziato in blocchi, nonchè da un database di stato per mantenere lo stato corrente del fabric. Per ogni canale c'è un registro e di conseguenza, ogni peer mantiene una copia del registro per ogni canale di cui è membro. Alcune caratteristiche del ledger:

- Query e aggiornamento del registro utilizzando ricerche basate sulla chiave, query di intervallo e query di chiavi composite
- Le transazioni consistono in versioni di chiavi / valori che sono stati letti in chaincode (set di lettura) e chiavi / valori scritti in chaincode (set di scrittura)
- Le transazioni contengono le firme di ogni peer atto all'approvazione
- Le transazioni vengono ordinate in blocchi e vengono trasmesse ai peer nel canale

- I peer convalidano le transazioni con le politiche di approvazione e applicano le policy stabilite
- Prima di inserire un nuovo blocco, viene eseguito un controllo della versione per garantire che gli stati delle risorse che sono state lette non siano cambiati dal momento dell'esecuzione del codice chaincode
- La transazione validata ed eseguita, è immutabile. Per caratteristica stessa del ledger
- Il ledger di un canale contiene un blocco di configurazione che definisce le policy, le liste per il controllo dell'accesso e altre informazioni simili
- I canali contengono istanze Membership Service Provider

Privacy Hyperledger Fabric utilizza un ledger immutabile per ogni canale, assieme ad una chaincode che può modificare lo stato degli asset (ad esempio, aggiornare la coppia chiave-valore). Un ledger esiste nel contesto di un canale; può essere condiviso nella rete (assumendo che tutti i partecipante siano operativi in un canale comune), o può essere reso privato e includere solamente un determinato set di partecipanti. In questo contesto, i partecipanti creano un canale separato e, dunque, isolano e limitano le loro transazioni ed il ledger. In determinati scenari in cui si desidera una connessione tra la trasparenza totale e la privacy, la chaincode può essere installata solamente sui nodi che necessitano di accedere allo stato degli asset per le operazioni di lettura e scrittura; in altre parole, se la chaincode non è installata su un nodo, non sarà in grado di interfacciarsi in maniera completa con ledger. Quando un sottoinsieme di organizzazioni su uno specifico canale ha necessità di mantenere i dati delle loro transazioni confidenziali, una collection privata di dati viene utilizzata per conservare questi dati in un database privato, logicamente separato dal ledger del canale, accessibile solo dall'insieme autorizzato delle organizzazioni. Dunque, i canali mantengono private le transazioni dal resto della rete, mentre le collection mantegono i dati privati all'interno di sottoinsiemi di organizzazioni nel canale. Per nascondere ulteriormente i dati, i valori dentro la chaincode possono essere cifrati (parzialmente o totalmente) usando algoritmi crittografici standard, come AES, prima di inviare le transazioni e aggiungere blocchi al ledger. Una volta che i dati cifrati sono stati scritti nel ledger, possono essere decifrati solamente dall'utente che possiede la chiave corrispondente usata per generare il testo cifrato.

Sicurezza e Servizi di Membership Hyperledger Fabric si basa su una rete composta da transazioni nella quale tutti i partecipanti hanno un'identità nota. La "Public Key Infrastructure" viene utilizzata per generare certificati crittografici che

sono legati alle organizzazioni, componenti della rete e utenti finali o applicazioni client. Conseguentemente, il controllo dell'accesso ai dati può essere gestito sia sull'intera rete, sia a livello di canale. Questo concetto di autorizzazione, unito a quello di canale con tutte le sue caratteristiche, permette di gestire situazioni in cui la privacy e la confidenzialità sono di interesse primario.

Consenso Nella tecnologia basata sul ledger distribuito, il consenso è stato recentemente associato ad un algoritmo specifico, che svolge una sola funzione. Tuttavia, il consenso svolge un'operazione più importante del semplice accordo per ordine delle transazioni: questa differenziazione è marcata in Hyperledger Fabric attraverso il suo ruolo principale all'interno della transazione, a partire dalla proposta per finire con l'approvazione, passando per l'ordinamento, la validazione e l'esecuzione. Brevemente, il consenso è una verifica completa della correttezza di un set di transazioni riguardanti un blocco. Il consenso viene raggiunto una volta che l'ordine ed i risultati delle transazioni di un blocco hanno soddisfatto determinato controlli dei criteri di policy. Questi controlli e bilanciamenti avvengono durante il ciclo di vita di una transazione, e includono l'utilizzo di policy di approvazione per stabilire quali membri devono approvare determinate transazioni, così come l'utilizzo di chaincode di sistema per garantire che queste policy siano state applicate e confermate. Prima dell'esecuzione, i nodi devono attuare queste chaincode di sistema per assicurarsi di aver accumulato approvazioni sufficienti, e che siano state date dalle opportune entità. In più, verrà effettuato un controllo di versione, durante il quale viene accordato o acconsentito lo stato corrente del ledger, prima che qualsiasi blocco contenente transazioni sia aggiunto al ledger. Questo controllo finale permette di evitare operazioni ridondanti e altre minacce che possano compromettere l'integrità dei dati, e permette ad alcune funzioni di essere eseguite su variabili non statiche. In aggiunta alle approvazioni, validità e controllo delle versioni, vengono effettuate verifiche di identità in tutte le direzioni nel processo della transazione. Le liste di controllo dell'accesso sono implementate su livelli gerarchici nella rete, e i pacchetti sono ripetutamente firmati, verificati ed autenticati non appena una proposta di transazione attraversa i differenti componenti dell'architettura. Il consenso, in conclusione, non si limita a concordare l'ordine di alcune transazioni; piuttosto, è una caratteristica generica e fondamentale ottenuta come risultato di una serie di verifiche che vengono attuate durante tutto il processo che riguarda la transazione, dalla proposta fino all'esecuzione.

Composer

Ulteriore framework utilizzato dell'ecosistema Hyperledger è Composer. Lo scopo di questo tool è quello di accelerare il processo di sviluppo e rendere più semplice l'integrazione delle applicazioni blockchain in sistemi di business già esistenti. Il

Composer permette di modellare una business network ed integrare sistemi e dati esistenti nell'applicazione blockchain. Hperledger Composer supporta le infrastrutture blockchain di Hyperledger Fabric già esistenti, che supportano i protocolli di consenso per assicurare che le transazioni siano validate, secondo le policy dei partecipanti della business network.
Hyperledger Composer è un modello di programmazione che comprende un linguaggio di modellazione, un set di API per definire e sviluppare business network e applicazioni che permettono ai partecipanti di inviare transazioni per scambiare asset.

Storge di Stato della Blockchain Tutte le transazioni all'interno della business network sono immagazzinate nel ledger della blockchain, e lo stato corrente degli asset e dei partecipanti è contenuto all'intero del database di stato della blockchain. Il ledger e lo stato vengono distribuiti attraverso un insieme di peer e viene garantito che qualsiasi aggiornamento sia consistente per tutti i peer che utilizzano un algoritmo di consenso.

Connection Profiles Il Composer utilizza i Connection Profiles per definire il sistema al quale con- nettersi. Un profilo di connessione è un documento JSON che fa parte della business network card. Questi profili, generalmente, sono forniti dal creatore del sistema e dovrebbero essere utilizzati per creare le business network card, in modo da poter accedere al sistema.

Asset Gli asset anche in questo caso sono elementi tangibili o intangibili, servizi o proprietà, e sono immagazzinati in registri. Gli asset possono potenzialmente rappresentare qualsiasi cosa all'interno di una business network, documenti, oggetti reali o brevetti. L'asset deve avere un identificativo unico, ma possono contenere qualsiasi proprietà sia necessario definire e inoltre possono essere collegati ad altri asset o partecipanti.

Partecipanti I partecipanti sono i membri di una business network. Possono possedere gli asset ed eseguire transazioni. Il tipo di partecipante viene modellato e possiede un identificativo unico e può contenere qualsiasi tipo di proprietà richiesta.Vi è la possibilità che un partecipante può essere mappato con uno o più identificativi.

Identità Un'identità consiste in un certificato digitale ed una chiave privata. Le identità vengono usate per eseguire le transazioni su una business network e dev'essere mappata ad un partecipante. La business network card contiene una singola identità e se questa è stata mappata ad un partecipante, permette

al possessore della business network card di eseguire transazioni nella business network.

Business Network cards Una Business network card è composta da un'identità, un profilo per la connessione, e metadati; questi ultimi, opzionalmente, contengono il nome della business network alla quale connettersi. Questa card semplifica il processo di connessione alla business network, ed estende il concetto il concetto di identità all'esterno della rete in un "contenitore", nel quale ciascuna è associata ad una specifica business network e ad un profilo per la connessione.

Transazioni La transazione è un meccanismo attraverso cui i partecipanti interagiscono con gli asset. Questo meccanismo implica la modifica delle proprietà dell'asset implicato, ad esempio il passaggio di proprietà da un partecipante all'altro.

Query Le query restituiscono dati che riguardano lo stato della blockchain. Sono definite all'interno di una business network, e possono essere parametrizzate con l'utilizzo variabili. Attraverso l'utilizzo delle query, i dati possono essere facilmente estratti dalla rete, e sono eseguite direttamente tramite l'API del Composer.

Eventi Gli eventi sono descritti nella definizione della business network, così come gli asset ed i partecipant. Una volta che gli eventi sono stati modellati, una transazione può comunicare ai sistemi esterni che è avvenuto qualcosa di significativo riguardante il ledger. Le applicazioni possono esser sottoscritte agli eventi emessi attraverso l'API del composer-client.

Controllo dell'accesso La business network potrebbe contenere un set di regole per il controllo dell'accesso. Queste regole permettono un controllo con granularità fine per quanto riguarda l'accesso che i partecipanti possiedono sugli asset nella business netowrk e le condizioni da rispettare. Il linguaggio che determina il controllo all'accesso è sufficientemente avanzato da permettere specifiche condizioni.

Cronologia La cronologia è un registro specializzato che contiene le transazioni eseguite con successo, includendo i partecipanti e le identità che l'hanno effettuata. Le transazioni vengono registrate come asset nominate HistorianRecord, che sono definite nel namespace del Composer.

Hyperledger Fabric SDK

L'SDK offre un layer di astrazione usato da applicazioni client per interagire con la rete blockchain di Hyperledger Fabric.

In particolare, l'utilizzo delle SDK Java permette alle applicazioni Java di gestire il ciclo di vita dei canali Hyperledger e anche la chaincode dello user. In particolar modo la grandissima utilità di questa SDK è poichè mette a disposizione tantissime funzionalità per l'esecuzione della chaincode dell'utente, blocchi di query, transazioni preconfezionate sul canale e monitor per gli eventi del canale. Per questi motivi si è fatto gran uso nella realizzazione dello strato di ledger di DME.

Tuttavia l'SDK non provvede dei mezzi di persistenza per le applicazioni, ma questo per scelta poichè lascia la gestione alla creazione di canali che possono essere serializzati attraverso la serializzazione di Java. Quindi l'applicazione che utilizza queste SDK necessita della realizzazione di politiche di gestione per la migrazione di dati.

Certificate authority Di notevole rilevanza è il fatto che l'SDK provvede una certificate authority, in modo tale che il client possa comunicare in maniera sicura con il layer ledger. Comunque l'SDK non è dipendente da un particolare implementazione di una certificate authority, infatti possono essere usate altre implementazioni di certificate authority. Questa è stata veramente importante per la realizzazione dell'interfaccia di registrazione dell'utente all'interno della blockchain.

SDK Gateway Di largo utilizzo è stato anche il Fabric Gateway SDK. Questo ha permesso di agganciarsi alla rete di blockchain e ha permesso all'applicazione di interagire con la stessa. Tutto questo grazie a delle semplici API che consentono di fare il submit delle transazioni al ledger o di fare delle query ai contenuti sempre del ledger veramente con un codice minimale. I principali elementi utilizzati del Fabric Gateway sono stati:

- **Wallet** che definisce l'interfaccia per memorizzare e gestire l'identità degli utenti nella rete di Fabric.
- **Gateway** mediante l'utilizzo del metodo *createBuilder()* configura la connessione usata per l'accesso al gateway impostando il wallet e la configurazione della rete.

Dipendenze dell'SDK L'SDK dipende da librerie di terze parti che per un corretto funzionamento della stessa deve essere incluso nel classpath quando si usano i JAR.
In particolare per buildare il progetto si deve aggiungere:

- Una versione della JDK 1.8 o una versione superiore;

- una versione di Apache Maven 3.5.0

Poi per lanciare la CA di Fabric o per gli Unit test che in DME non sono stati realizzati si deve integrare:

- La versione di Docker 18.03;
- Docker Compose 1.21.2

5.5.2 Linux

Lo sviluppo e il deploy dello strato di ledger è stato completamente realizzato in ambiente Linux mediante Virtual Machine.
Linux è una famiglia di sistemi operativi open source di tipo Unix-like, pubblicati in varie distribuzioni, aventi la caratteristica comune di utilizzare come nucleo il kernel Linux. Inizialmente per necessità per compatibilità, poichè le prime fasi sperimentali dell'utilizzo di Hyperledger sono state realizzate mediante l'utilizzo del Composer di Fabric, si è utilizzata una versione specifica di Linux: la distribuzione di Ubuntu con la versione specifica 16.01.
Successivamente alla fase di sperimentazione e alla scelta implementativa di realizzare lo strato del ledger mediante l'utilizzo del solo Fabric congiunto alle SDK, si è continuato ad utilizzare sempre un ambiente Linux Ubuntu poichè vi erano già presenti molti script pronti per la gestione e inizializzazione della chaincode eseguibili solo in questo ambiente.

Virtual Machine I servizi in fase di sviluppo giravano in locale, per sviluppare il layer distributed in Linux si è fatto utilizzo di una Virtual Machine, in particolare Oracle Virtual Box.
VirtualBox è un software di virtualizzazione x86 e AMD64 / Intel64 per uso aziendale e domestico. VirtualBox non è solo un prodotto estremamente ricco di funzionalità e ad alte prestazioni per i clienti aziendali, inoltre è anche l'unica soluzione professionale che è disponibile gratuitamente come software Open Source secondo i termini della GNU General Public License (GPL) versione 2.
[5] Una volta virtualizzato il sistema operativo Ubuntu è stato necessario configurare il tipo connessione alla scheda di rete per riuscire a comunicare in locale con il servizio che girava sull'ambiente virtualizzato. Nello specifico si è dovuto cambiare il collegamento della scheda da *NAT* a *scheda con bridge*. Successivamente con il comando *ifconfig* si è preso l'indirizzo locale della macchina e la sua porta che è servito per comunicare con essa dal nostro Front End.

5.5.3 Spring

Spring Framework fornisce un modello completo di programmazione e configurazione per le moderne applicazioni aziendali basate su Java, su qualsiasi tipo di piattaforma di distribuzione.
Un elemento chiave di Spring è il supporto infrastrutturale a livello di applicazione: Spring si concentra sul "plumbing" delle applicazioni aziendali in modo che i team, proprio come nel nostro caso, possano concentrarsi sulla logica di business a livello di applicazione, senza legami inutili con ambienti di distribuzione specifici.
[6] In particolare, l'obiettivo era quello di rendere il distributed Ledger disponibile all'interno di un servizio Web RESTful come gli altri applicativi e Spring era proprio il framework adatto al nostro scopo.

Spring Boot Tra le varie scelte che Spring mette a disposizione la particolare distro di Spring utilizzata è stata lo Spring Boot, le cui caratteristiche principali sono:

- Incorpora direttamente Tomcat, Jetty o Undertow (non è necessario distribuire file WAR);
- Fornisci delle dipendenze "iniziali" per semplificare la configurazione della build;
- Configura automaticamente le librerie Spring e di terze parti quando possibile;
- Fornire funzionalità pronte per la produzione come metriche, controlli di integrità e configurazione esternalizzata
- Nessuna generazione di codice e nessun requisito per la configurazione XML

Capitolo 6

Fase sperimentale e sviluppo dello Smart Contract

6.1 Fasi preliminari

6.1.1 Scelta della tecnologia

La scelta di utilizzare Fabric, nella sua versione pura non è stata immediata.
In prima battuta è stato valutato l'utilizzo del Composer di Fabric, in quanto garantiva,da come è riportato dalle documentazioni, dei vantaggi in termini di sviluppo.
Come detto in precedenza Composer permette uno sviluppo più veloce in quanto rende più semplice l'integrazione e l'implementazione di una blockchain.

Installazione prerequisiti Composer Per l'implementazione della blockchain si è preso come decisione di svilupparla su un ambiente Linux. Quindi di conseguenza per installare i prerequisiti di Composer si sono dovuti installare tutti gli strumenti necessari per sviluppare in questo sistema operativo, che nello specifico è la versione Ubuntu 16.04 LTS.
Gli strumenti che sono stati installati sono i seguenti:

- Docker Engine: Version 17.03;
- Docker-Compose: Version 1.8;
- Node: 8.9;
- Npm: v5.1;
- Git: 2.9;

- Python: 2.7;

L'installazione dapprima intrapresa singolarmente per ogni strumento potrebbe far sorgere alcuni problemi, dunque se si è scelto proprio la versione di Linux Ubuntu 16.04 LTS, esiste uno script messo a disposizione dalla documentazione ufficiale di Hyperledger Fabric Composer che permette l'installazione automatica di tutti questi strumenti.
Per l'esecuzione dello stesso script vengono qui riportati i comandi utili per eseguirlo:
-curl -O https://hyperledger.github.io/composer/v0.19/prereqs-ubuntu.sh

-chmod u+x prereqs-ubuntu.sh

Installazione e configurazione Composer Di seguito si è installato una serie di strumenti client che sono utili per gli svuluppi tramite Composer e gli strumenti utili per eseguire il server Rest sulla macchina come API RESTful.
Lo strumento CLI essenziale lo si installa con il comando seguente:
-npm install -g composer-cli@0.20
Mentre per installare il server Rest:
-npm install -g composer-rest-server@0.2 Molto utile in ambiente composer è un applicazione browser che serve per modificare,gestire e modificare la business network, questa si chiama Playground e può essere eseguito in locale durante lo sviluppo. Per installarlo:
-npm install -g composer-playground@0.20

Installazione Fabric Unita all'installazione di questi componenti è fondamentale l'installazione dell'Hyperledger Fabric. Questa fornisce un insieme di comandi che servono per installare un runtime locale di Fabric.
La documentazione ufficiale di Fabric mette a disposizione uno script per il download dello stesso. Per scaricarlo ed eseguirlo, riportiamo i comandi che sono stati lanciati:
- curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz
- tar -xvf fabric-dev-servers.tar.gz Ci si deve riportare sulla directory dove si è scaricato lo script ed eserguirlo con:
- /.downloadFabric.sh

Sperimentazione di utilizzo di una soluzione Composer Prima di un'eventuale implementazione di una nostra soluzione per DME, è stato valutato e sperimentato l'utilizzo del Composer sfruttando le demo messe a disposizione di

Hyperledger stesso.
Lo script di cui si è disposizione dopo aver fatto il download, è molto utile in quanto non solo permette di fare questo tipo di valutazione in fase sperimentale ma è utilissimo anche in fase di sviluppo come vedremo successivamente.
Lo script *./startFabric.sh* si occupa dell'installazione e della configurazione non integrate nella rete stessa. Oscura le basi della rete a quel livello e questo va bene per la maggior parte degli sviluppatori di applicazioni. Non devono necessariamente sapere come funzionano effettivamente i componenti di rete in dettaglio per creare la loro app. In particolare nella sua configurazione di default:

- crea un canale e unisce il peer al canale
- installa lo fabcar smart contract sul file system del peer e lo istanzia sul canale (l'istanza avvia un contenitore)
- chiama la initLedger funzione per popolare il registro del canale con 10 auto uniche

Cioè che viene inizializzato consiste in un singolo nodo peer configurato per utilizzare CouchDB come database di stato, un singolo nodo, un'autorità di certificazione (CA) e un contenitore CLI per l'esecuzione dei comandi.
Come secondo passo eseguito, viene eseguito lo script per creare una PeerAdminCard che ricordiamo contiene una singola identità e viene mappata ad un partecipante, in questo caso l'admin, e permette al possessore della business network card di eseguire transazioni nella business network.
Successivamente è stato utilizzato un tool per aiutarci nella generazione dell'applicazione, lo Yeoman (*npm install -g yo*), che in particolare è servito per generare la struttura della BND, la quale definisce il data model, la logica delle transazioni e le regole per il controllo dell'acceso per la blockchain. Per far ciò si deve specificare nei comandi *yo* il nome della business network, il nome e la mail dell'autore, il tipo di server, il namespace e specificando che non si vuole generare una una BND vuota avremo uno scheletro composto da asset, partecipant,transaction e le regole di access control, query ed event.
Il modello viene rappresentato in un file ".cto" che contiene le definizioni delle classi per tutti gli asset, partecipant e transaction nella rete. Contiene anche un documento per l'access control (permissions.acl) con le regole di base, uno script (logic.js) che contiene le funzioni per eseguire le transazioni,e il file package.json che contiene i metadata della business network.
Ai fini di sperimentazione della piattaforma, dunque è stato modificato il model aggiungendo un partecipant "Trader", un asset "Commodity"e una transazione "Trade"che inidica la relazione tra Trader e Commodity.
Il file logic.js descrive le funzioni per le transazioni contiene la logica JavaScript per eseguire le transazioni definite nel model file. La transazione "Trade" è stata

definita in modo tale che accetti l'identificativo dell'asset "Commodity" che deve essere scambiato, e l'identificativo del partecipant "Trader" per impostare un nuovo possessore.
Il successivo passo eseguito è stato creare una business network archive per farle

```
/**
* Track the trade of a commodity from one trader to another
* @param {org.example.mynetwork.Trade} trade - the trade to be process
* @transaction
*/
async function tradeCommodity(trade) {
    trade.commodity.owner = trade.newOwner;
    let assetRegistry = await getAssetRegistry('org.example.mynetwork.
    await assetRegistry.update(trade.commodity);
}
```

Figura 6.1: Codice relativo alla funzione della transazione

contenere la BDN e infine fare il deploy della business network. Normalmente sono richieste le informazioni dell'amministratore per creare un'identità PeerAdmin, con i privilegi per installare la chaincode sul peer e lanciarla sul canale. L'identità PeerAdmin è stata creata durante la fase di installazione dell'ambiente di sviluppo. Il deploy della business network in Hyperledger Fabric richiede che sia installata sul peer, dopodiché può essere lanciata, e un nuovo partecipant, l'identity e la relativa card devono essere create dall'amministratore di rete. Infine, la business network card dell'amministratore di rete deve essere importata ed è possibile effettuare il ping per verificare che la rete sia attiva.

Motivazioni per cui si è deciso di abbandonare Composer Nonostante i vantaggi dell'uso di Composer siano davvero notevoli, come si evince anche da questa fase di sperimentazione, è stata fatta un'attenta analisi prima di procedere gli sviluppi con questa tecnologia.
In generale secondo lo stato attuale dell'arte, Composer è stato considerato come deprecato. Perché però è considerato tale se i vantaggi sono molti e variegati?
In sostanza è che la maggiori multinazionali come IBM, che sta investendo molto sulle blockchain, lo hanno abbandonato. Dunque di conseguenza se le multinazionali abbandonano questa pista, di conseguenza si muove tutto il mercato. Delle motivazioni che fornisce IBM di questa scelta possono essere riassunte in:

- Composer è stato progettato fin dall'inizio per supportare più piattaforme blockchain, non solo Fabric, ma questo design ha un costo. Questo design ha fatto sì che esistano due modelli di programmazione completamente diversi: il modello di programmazione Fabric (chaincode) e il modello di programmazione Composer (reti aziendali). Ciò ha causato una notevole confusione agli utenti, con la necessità di fare una "scelta" tra i due modelli di programmazione, con pochissime somiglianze tra i due. In questo caso particolare, la scelta è stata una brutta cosa, con molti utenti che hanno scelto di non utilizzare la parte "opzionale" dopo l'esplorazione iniziale o la fase POC.
- Questo design ha anche reso molto più difficile per IBM adottare ed esporre le ultime funzionalità di Fabric. é estremamente difficile per IBM stare al passo e allineati con le ultime funzionalità di Fabric. Ciò ha significato che gli utenti hanno comprensibilmente smesso di utilizzare Composer e invece sono tornati a sviluppare con Fabric.

Infine il confronto con Linear ci ha spinto ulteriormente a prendere la scelta di cominciare lo sviluppo semplicemente utilizzando una soluzione che prevede l'utilizzo del solo Hyperledger Fabric.

6.2 Sviluppo chaincode/ Smart Contract

6.2.1 Inizializzazione della rete

Per quanto riguarda l'inizializzazione della rete, il nostro applicativo DME non richiede una particolare configurazione della rete blockchain, si è scelto dunque di utilizzare lo script di inizializzazione di default di Fabric: */startFabric.sh*. Questo comando creerà una rete blockchain composta da due peer e im' autorità di certificazione . Inoltre installerà e istanzierà una versione TypeScript del nostro contratto smart DME che verrà utilizzata dalla nostra applicazione per accedere al ledger. Dunque si è solo configurato l'avvio all'interno della rete della nostra particolare chaincode. Lo schema della rete secondo il quale dovrebbe operare DME può essere riassunto con uno scema riportato dalla documentazione di Hyperledger:

6.2.2 Stesura dello Smart Contract relativo a DME

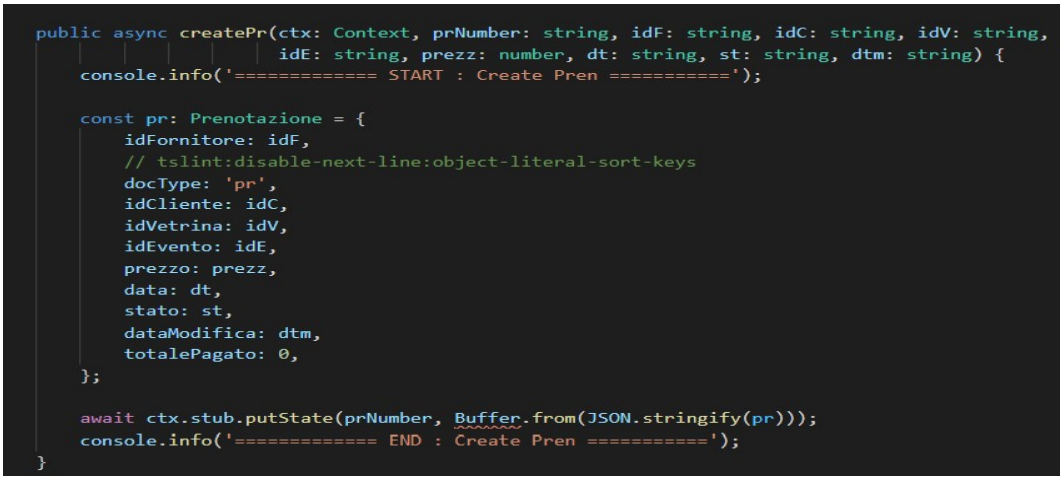
Prima di tutto si è dovuto scegliere in quale linguaggio sviluppare la chaincode. Hyperledger mette a disposizione la possibilità di scegliere tra diversi tipi di linguaggi per la scrittura dello Smart Contract, tra questi: GO, JavaScript,TypeScript, C++. Poichè in termini di prestazioni o difficoltà non c'è nessun vantaggio di uno rispetto all'altro, la scelta è ricaduta sul TypeScript solo per una motivazione di confidence con il linguaggio stesso.

Figura 6.2: Rete blockchain DME

Si è utilizzato l'Hyperledger Contract API, estendendo la classe **Contract** con la nostra classe DMEContract. All'interno sono stati implementati i metodi asincroni che fanno parte della logica del contratto, questi verranno elencati di seguito:

- **initLedger**: prende come argomento il Context, un'altra Hyperledger Contract API che rende disponibile il contesto transazionale per l'esecuzione delle transazioni. L'initLedger da come si evince dal nome stesso, inizializza il registro, in particolare ricostruendo le Prenotazioni(che vedremo successivamente, sarà l'elemento che verrà trattato e registrato all'interno del ledger) all'interno del nostro ledger
- **createPr**:questo metodo viene utilizzato, passando tutti i parametri necessari per la Prenotazione, per la creazione di una nuova prenotazione e l'inserimento nel ledger, sfruttando sempre il metodo del context: *ctx.stub.putState(prNumber, Buffer.from(JSON.stringify(pr)))*
- **updatePr**:metodo analogo al precedente, solo che aggiorna una prenotazione esistente all'interno del ledger. Si ricorda che la transazione effettivamente aggiorna in questo caso la prenotazione, ma il registro mantiene traccia di tutte le transazioni.
- **getPrHistory**:questo metodo restituisce tutte le transazioni legate ad una prenotazione per ricostruirne uno storico, sfruttandone appunto la caratteristica del ledger che tiene memoria di transazione per transazione
- **queryAllPrenotazioni**:metodo di utility in caso fosse necessario restituire tutte le transazioni del ledger

Esempio dimostrativo di codice del metodo *createPR*: Dall'immagine appena



```
public async createPr(ctx: Context, prNumber: string, idF: string, idC: string, idV: string,
                      idE: string, prezz: number, dt: string, st: string, dtm: string) {
    console.info('============= START : Create Pren ===========');

    const pr: Prenotazione = {
        idFornitore: idF,
        // tslint:disable-next-line:object-literal-sort-keys
        docType: 'pr',
        idCliente: idC,
        idVetrina: idV,
        idEvento: idE,
        prezzo: prezz,
        data: dt,
        stato: st,
        dataModifica: dtm,
        totalePagato: 0,
    };

    await ctx.stub.putState(prNumber, Buffer.from(JSON.stringify(pr)));
    console.info('============= END : Create Pren ===========');
}
```

Figura 6.3: Metodo creazione prenotazione

portata si nota come la classe prenotazione giochi un ruolo fondamentale. Come già detto è l'elemento che viene registrato nel ledger e mantiene le informazioni necessarie per la logica di funzionamento della nostra DLT.
La struttura di *Prenotazione* è definita in un opportuno prenotazione.ts e importato all'interno della chaincode. I campi fondamentali di tale struttura sono:

- idCliente: memorizza l'id del cliente che compie la transazione;
- idVetrina: l'id della vetrina sulla quale il cliente effettua la prenotazione, che è legato al fornitore.
- idEvento: lega la prenotazione all'evento del cliente
- prezzo: il totale da saldare relativo alla fornitura del servizio da rispettare da parte del cliente
- data: data di consegna della fornitura, da rispettare da parte del fornitore
- stato: tiene traccia nel ledger dello stato della prenotazione effettuata. Una prenotazione può trovarsi nello stato di solamente "prenotato" oppure con un saldo non totalmente pagato, nel caso di pagamento a rate
- totalePagato: tiene appunto traccia di quanto il cliente ha pagato. Fondamentale per il vincolo di adempimento del contratto da parte del cliente. **Solo** quando il totale pagato coincide con il prezzo il cliente ha adempito al contratto.

- dataModifica: indica l'ultimo cambiamento di stato del Contratto in essere

Dunque questo è lo Smart Contract la cui istanza verrà inizializzata in fase di esecuzione della rete quando viene lanciata la blockchain all'interno dell'ambiente che la ospiterà.

6.3 Microservizio per lo Smart Contract

Per rendere disponibile lo smart contract all'utente finale per far si che possa fare le sue transazioni, si è implementato un microservizio in Spring Boot, che mediante l'esposizione di chiamate API RESTful si espongono le SDK di Hyperledger Fabric per interagire con la blockchain.

6.3.1 Servizi

I servizi che sono stati implementati a supporto sono:

- **ChainCodeService**
- **RegisterService**

RegisterService Il servizio di registrazione viene utilizzato all'atto della creazione di un nuovo cliente/fornitore. In fase di registrazione il FE oltre a chiamare l'endpoint relativo al salvataggio nel database del nuovo utente,mediante l'opportuno controller, che verrà mostrato in seguito, verrà chiamato il servizio di registrazione all'interno della blockchain.
Questo servizio, riceverà l'id dell'utente e lo salverà all'interno del wallet relativo alla nostra chaincode. In breve il wallet, è un container della SDK dove viene registrato una coppia chiave, valore, dove la chiave è l'id dello User mentre il valore è un' *Identity*. Questa Identity viene formata utilizzando un altro oggetto chiave l'HFCAClient. I passaggi sono i seguenti:

- Si crea una nuova RegistranRequest passando l'id dell'utente;
- Di questo nuovo oggetto, si settano: l'affilizione, ovvero il department della nostra blockchain, l'enrollement ID cioè quello dell'utente
- Chiamiamo la funzione register dell'oggetto HFCAClient, passando la registrationRequest e l'admin, questa ritornerà un enrollment secret
- Successivamente si passa alla enroll dell'HFCAClient l'id e l' enrollmentSecret e questa restituirà un oggetto Enrollment.

- Infine viene creata una nuova x509Identity passando l'enrollment, con un metodo dell'oggetto Identities che contiene il certificato dell'admin

- Infine viene aggiunto al wallet

```
RegistrationRequest registrationRequest = new RegistrationRequest(id);
registrationRequest.setAffiliation("org1.department1");
registrationRequest.setEnrollmentID(id);

String enrollmentSecret = caClient.register(registrationRequest, admin);
Enrollment enrollment = caClient.enroll(id, enrollmentSecret);
Identity user = Identities.newX509Identity("Org1MSP", enrollment);
wallet.put(id, user);
```

Figura 6.4: Register service

Questi due oggetti su citati vengono inizializzati come Bean Java all'avvio del servizio mediante sempre opportune SDK. Per utilizzarli all'interno del service viene fatta l'injection dei due oggetti.

ChainCodeService Il core dei servizi viene implementato in questa classe. Il metodo fondamentale che viene richiamato in ogni metodo é il *Contract connect(String user).*
Questo metodo utlizza l'SDK fondamentale di Hyperledger: il Gateway. Gateway ci permette di effettuare le operazioni primarie per agganciarci alla nostra blockchain. Prima di tutto verifica la presenza dello user all'interno del wallet che ne garantisce l'iscrizione non solo nella piattaforma, ma anche all'interno della blockchain. Passando le configurazioni della nostra rete(la peer organization, il nome della rete, il nome della connessione) al gateway builder lo inizializza per far si che a sua volt, ricevendo il nome del canale dentro il quale è inizializzato lo smart contract DME, crea un oggetto Network dell'SDK. Questo infine permette con il metodo getContract di restituire il nostro smart Contract DME.
Ognuno degli altri metodi servono per richiamare i metodi di creazione, aggiornamento, e restituzione della history.
Prendendo come spunto il metodo del service per a creazione della prenotazione: si nota che viene richiamato il metodo connect() precedentemente descritto, e successivamente viene fatta la submit della transazione sul contratto creando una nuova prenotazione e impostando lo stato su "creata". I dati sono prelevati da un DTO per la transazione che sono gestiti e passati alla *createPR* del contratto.

```
Gateway.Builder builder = Gateway.createBuilder();
builder.identity(wallet, user).networkConfig(networkConfigPath).discovery(true);

// create a gateway connection
Gateway gateway = builder.connect();

// get the network and contract
Network network = gateway.getNetwork("mychannel");

return network.getContract("DME");
```

Figura 6.5: Gateway

```
@Override
public void createTransaction(TransactionDTO dto, String user) {

    try {
        Date dtm = new Date();
        Contract contract = connection(user);
        contract.submitTransaction("createPr", dto.getTransactionId(), dto.getIdFornitore(), dto.getIdCliente(),
                dto.getIdVetrina(), dto.getIdEvento(), dto.getPrezzo().toString(), dto.getData(), "creata", dtm.toString());

    } catch (IOException | InterruptedException | TimeoutException e) {
        e.printStackTrace();
        throw new GenericException();
    }catch (ContractException e){
        e.printStackTrace();
        throw new ClientException();
    }
}
```

Figura 6.6: creazione della transazione

6.3.2 Controller

I controller espongono gli endpoint per servire le richieste provenienti dal FE, chiamano i servizi mediante injection degli stessi.

RegistrationController Endpoint mappato su "/registration", dispone di un unico metodo Post createUser per la creazione di un nuovo utente.
Viene passato mediante il RequestBody l'id dell'utente, che sarà creato al momento della creazione di un nuovo utente. Viene sfruttata la circostanza che questo id sarà sicuramente univoco, in quanto l'univocità dello stesso viene gestita dal DataBase di DME. Anche qui viene fatta l'injection del servizio di registrazione descritto in precedenza, il metodo di creazione viene così richiamato all'interno del post, e restituirà un 200 OK se la registrazione all'interno della blockchain è andata a buon fine.

TransactionController Endpoint mappato su "/transaction". Mediante l'injection della **chaincodeService** permette di utilizzarne tutti i suoi servizi, e quindi permettere di esporli come servizi Web, fondamentale per tutta la piattaforma, importanza che sarà discussa in seguito.
I metodi Post esposti sono i seguenti:

- **createTransaction:** Fondamentale per tutta la logica dello smart contract. Questo controller riceve come parametro lo userid dell'utente che sta prenotando un servizio e un dto con i parametri principali di questa transazione.
  Il controller utilizza il service della chaincode per creare all'interno della blockchain questo nuova prenotazione legata allo Smart Contract di DME. Di fondamentale importanza è la set dello stato della prenotazione, di deafault, all'atto della creazione viene impostato come "creata" e questo viene legato indissolubilmente alla prenotazione, al registro della blockchain e vale come stato del contratto. Per variare questa condizionare è possibile solo mediante un pagamento o uni interazione con il fornitore che agisce con la blockchain mediante altre chiamate.

- *transactionHistory:* Riceve come parametro lo userid e mediante questo dato chiama la getAllTransaction del service passando come argomento lo userid. Riceve la history delle transazioni legate ad una singola prenotazione di un servizio effettuate dal cliente, pronte per essere mostrate nel Front End.
  In questo modo l'utente può analizzare tutti i cambiamenti degli stati nel suo contratto con l'assoluta certezza offerta dalla blockchain e i suoi servizi, potendone osservare i vari cambiamenti di stato monitorando ora e data.

- *getTransaction:* Riceve come parametro lo userid e l'id della transazione. Utilizza il metodo della chaincode getTransactionById che permette di visualizzare le informazioni relative alla singola transazione, nel caso fosse necessario.

I metodi Get:

- *getAllTransactions:*questo metodo è un metodo di supporto. Riceve come parametro l'userid e restituisce tutte le transazioni, non più legate ad una prenotazione, ma una lista di tutte le transazioni legate al singolo utente.

I metodi Put:

- **updateTransaction:**Insieme al create Transaction questo è l'altro endpoint fondamentale dell'architettura di funzionamento della chaincode.
  Questo metodo aggiorna la prenotazione nei campi modificabili con una nuova transazione. Principalmente viene utilizzato per i cambiamenti di stato permessi per una determinata prenotazione.
  Si ricorda che tutte le transazioni e quindi i cambiamenti di stato della

prenotazione saranno visibili sia all'utente che al fornitore, stabilendo così un legame di trusting solido, grazie allo smart contract.

PagamentoController EndPoint mappato su "/pagamento". Anche questo controller utilizza il chaincodeService, e lo sfrutta per registrare e controllare gli stati del pagamento. Allo stato attuale dello sviluppo i pagamenti sono mockati, ma è lasciata una facile introduzione di un modulo che può implementare un servizio di pagamento reale, che può essere: un servizio bancario, PostePay, PayPal ecc.
L'unico metodo è un metodo Post: *paga*
Riceve come lo userid dell'utente che vuole effettuare il pagamento e come body la struttura dati Data Transfer Object della Prenotazione.
Il metodo prima di tutto controlla se il metodo di pagamento non sia nullo, in caso negativo viene ritornata una BAD_REQUEST al Front End. Successivamente segue la logica di gestione del pagamento e quindi delle condizioni di chiusura contrattuale. Mediante l'id dell'utente e della prenotazione si ricostruisce l'ultimo stato della prenotazione, da questo vengono trattati due dati fondamentali lo stato della prenotazione e l'importo totale pagato fino a quel momento. Dunque viene verificato se l'importo pagato in questa transazione più l'importo pagato fino a quel momento corrisponde all'importo totale del servizio prenotato. Da qui si aprono due scenari:

1. Se l'importo pagato più la somma dell'importo precedente è uguale al prezzo del servizio prenotato lo stato viene settato come **PAGATO** e di conseguenza viene aggiornato la data dell'ultimo pagamento e il totale pagato. A questo punto con una nuova transazione viene aggiornato tutta la prenotazione.

2. Se l'importo pagato più l'attuale pagamento non è uguale al totale all'importo del prezzo del servizio, questo viene considerato come una somma parziale dell'importo dovuto e quindi considerato come importo di pagamento di una rata.
   Lo stato quindi viene settato come **PAGAMENTO RATA** e viene aggiornato l'importo totale fino ad adesso pagato e come nel precedente caso la data dell'ultimo pagamento.

6.4 Visione d'insieme del funzionamento

In conclusione si può dire che la logica di funzionamento di tutto lo smart contract è distribuita tra i vari servizi, i controller,la chaincode e la rete.
Per la nostra infrastruttura si può dire che la rete mette a disposizione lo scheletro della blockchain, sfruttando l'avvio standard di una rete di Hyperledger che instanzia la nostra chaincode. Questa definisce il comportamento dello smart contract, ovvero

le possibili transazioni su quello che è l'asset cioè la Prenotazione.
I servizi mediante l'SDK comunicano con la blockchain e immettono all'interno del registro della blockchain le varie transazioni.
Infine i controller attivano le varie funzionalità dei servizi e sono attivati dal front end in quanto espongono degli endopoint che sono contattati dal web. In più implementa delle logiche di validazione dei vari stati o il pagamento.
Da sottolineare il fatto che eventuali anomalie non possono sicuramente presentarsi all'interno della blockchain quanto è praticamente inattaccabile in nessun modo. Ma in aggiunta eventuali anomalie di inserimento, o tentativi di evasione ad adempimento del pagamento verranno comunque registrati dunque al limite sia fornitore che cliente sono tutelati sotto questo punto di vista.

Capitolo 7

Conclusioni

7.1 Percorso svolto

La fase preliminare del progetto DME è stata incentrata sull'analisi dei requisiti, analisi della user story e degli use case, definizione degli scenari di utilizzo.
Successivamente, è stato definito il design grafico dell'interfaccia utente con dei mockup elaborati tramite l'uso di Adobe XD.
Terminata la fase di definizione delle specifiche è iniziata la fase di design architetturale in cui è stata posta particolare attenzione sulla struttura del database, dei microservizi, logica frontend.
Lo sviluppo di DME, nel suo complesso, è durato circa sei mesi. Una volta terminati front end e back end è stato aggiunta la sezione dello smart contract con i relativi servizi e implementazione di chaincode.
Terminati gli sviluppi è inziata la fase di deploy sui server di proprietà di Linear System.

7.2 Sviluppi futuri

A seguito del lavoro che è stato svolto sino ad ora con Linear System è in fase di progettazione da parte dell'università di Bari un modulo recommender system.
Il modulo recommender system si occuperà di avvicinare in modo automatico e puntuale la domanda e l'offerta di servizi relativi agli eventi. In genere i sistemi di raccomandazione trovano applicazione in diversi settori, ed il loro scopo è quello di aiutare le persone ad effettuare scelte basandosi su diversi aspetti.
Data una persona, questi aspetti possono essere ad esempio la propria cronologia, ovvero gli acquisti già effettuati o i voti positivi già dati, o le preferenze di persone simili ad essa. In base a questi aspetti e a come sono prodotte le raccomandazioni

i sistemi di raccomandazione si dividono in varie tipologie.
I sistemi di raccomandazione possono essere raggruppati in tre categorie principali, in base all'approccio usato per ottenere le raccomandazioni, più una quarta che è nata in seguito alla crescente diffusione dei social network, e che viene enunciata per ultima:

- Approcci content-based: alla persona saranno raccomandati oggetti simili a quelli che ella ha preferito in passato;
- Approcci collaborativi: alla persona saranno raccomandati oggetti che persone con preferenze simili alle sue hanno preferito in passato;
- Approcci ibridi: questa tipologia racchiude i sistemi di raccomandazione che combinano tecniche usate nelle due precedenti tipologie;
- Approcci semantic-social: si considera un insieme di persone ed uno di oggetti, dove le persone sono connesse da una rete sociale e sia queste ultime sia gli oggetti sono descritti da una tassonomia.

7.3 Considerazioni finali

In base ai requisiti e alle specifiche del progetto sono state implementate e testate le seguenti funzionalità per la creazione e gestione di eventi:

- Registrazione del cliente/fornitore
- Autenticazione
- Creazione e modifica di un evento
- Creazione e modifica di un servizio
- Ricerca e prenotazione di un servizio
- Visione e prenotazione di appuntamenti
- Gestione degli invitati
- Gestione tavoli
- Creazione e gestione di smart-contract tramite l'uso di una blockchain

Questo progetto ci ha messo di fronte all'analisi e allo studio di tecnologie più utilizzate al momento nell'ambito dello sviluppo di applicazioni web. Il lavoro di tesi è stato molto formativo sia nell'applicare le conoscenze apprese durante

i corsi tenuti dal Politecnico di Torino, sia nell'apprendere nuove competenze e interfacciarsi con una realtà aziendale moderna.
Queste esperienze andranno ad arricchire il nostro bagaglio tecnologico e interpersonale rendendoci più preparati ad affrontare il mondo del lavoro.
Una delle sfide più grandi che ci hanno messo alla prova durante l'evoluzione del progetto è stata la modalità di lavoro in smart-working a causa della pandemia Covid-19, allo stesso tempo questo ci ha permesso di sfruttare tecnologie per l'organizzazione del lavoro di gruppo come Microsoft Teams, Git e Jira e ci ha spronato a gestire al meglio il nostro tempo in linea con le fasi di sviluppo.

Bibliografia

[1] RedHat. *Cosa sono i microservizi?* [Online; in data 24-febbraio-2021]. 2021. URL: https://www.redhat.com/it/topics/microservices/what-are-microservices (cit. alle pp. 4, 8, 9).

[2] Microsoft. *Stile di architettura di microservizi*. [Online; in data 24-febbraio-2021]. 2021. URL: https://docs.microsoft.com/it-it/azure/architecture/guide/architecture-styles/microservices (cit. alle pp. 5, 6).

[3] Ibm. *Cos'è la tecnologia blockchain?* [Online; in data 01-marzo-2021]. 202o. URL: https://www.ibm.com/it-it/topics/what-is-blockchain (cit. alle pp. 53, 54).

[4] Hyperledger. *Hyperledger fabric?* [Online; in data 03-marzo-2021]. 2020. URL: https://www.hyperledger.org/use/fabric (cit. a p. 59).

[5] Virtual box org. *Virtual box*. [Online; in data 06-marzo-2021]. 2020. URL: https://www.virtualbox.org/ (cit. a p. 66).

[6] Wikipedia. *Spring framework*. [Online; in data 06-marzo-2021]. 2020. URL: https://it.wikipedia.org/wiki/Spring_Framework (cit. a p. 67).