# POLITECNICO DI TORINO

**Corso di Laurea Magistrale in Ingegneria Informatica**


**Tesi di Laurea Magistrale**

# Smart Contract nell'organizzazione di eventi
# Front End

**Relatori**

**Prof. Luca ARDITO**

**Prof. Maurizio MORISIO**

**Candidato**

**Martino MASSA**

**Aprile 2021**

# Sommario

***Contesto*:** Le tecnologie basate su blockchain sono oggetto di studio e implementazione fino dalla loro prima applicazione nel sistema Bitcoin. Uno degli sviluppi maggiormente interessanti è rappresentato dalla contrattazione smart, in cui alcune clausole vengono tradotte in software per l'esecuzione automatica al verificarsi di una determinata condizione. Il sistema risolve alla radice il problema di eventuali inadempimenti, spostando l'intervento della tutela giudiziale a un momento successivo in cui verranno verificate le condizioni di validità ed efficacia del contratto o l'eventuale presenza di errori di sistema. In particolare gli smart contract verranno applicati alla contrattazione cliente-fornitore nel campo dell'organizzazione di eventi.

***Obiettivi*:** 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 propone di realizzare una piattaforma digitale di supporto alla organizzazione, gestione, comunicazione e partecipazione ad eventi. La piattaforma inoltre creerà e gestirà i contratti di fornitura dei diversi servizi attraverso degli smart contract, implementati in Hyperledger. Sarà quindi necessario prima di tutto sviluppare l'architettura a microservizi per il lato backend e il lato front end della piattaforma che permetterà la connessione tra clienti e fornitori, successivamente il front end verrà integrato con Hyperledger per la definizione e gestione di smart contract.

***Metodo*:** Il modulo front-end, sviluppato in Angular, ha la responsabilità di erogare servizi di accesso e rendere 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. La progettazione prevede semplicità, accesso multi-device e in mobilità. Il modulo backend è basato su architettura a microservizi in cui ciascun microservizio è composto di funzionalità elementari riusabili. L'implementazione di ciascun microservizio prevede l'utilizzo di Java, Spring e Hibernate per la realizzazione di market place di definizione di beni e servizi, definizione di utenti e profili, servizi orizzontali di autorizzazione e autenticazione,

back up e recovery. Infine, è stato realizzato, con l'utilizzo di Hyperledger Fabric, 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.

***Conclusioni*:** Attraverso la comunicazione tra tutti e tre i moduli descritti si è riusciti a soddisfare i requisiti iniziali del progetto ovvero tutte le funzionalità di supporto all' organizzazione, gestione, comunicazione e parteci-pazione ad eventi. Tuttavia, rimangono dei dubbi sull'effettiva validità legale degli smart contract in quanto per la legislazione presente non rispetterebbero i vincoli di forma, oggetto, propri di un contratto legale.

# 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 participare 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, 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 macroperiodi 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."

é 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

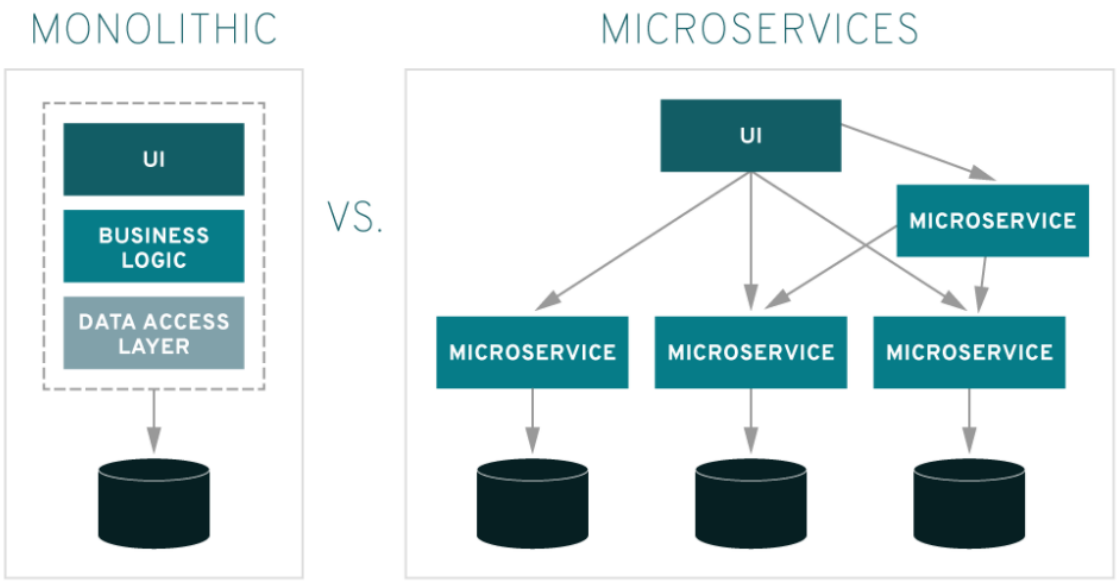



**Figura 2.1:** confronto tra schema monolitico e schema microservizi[1]

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.

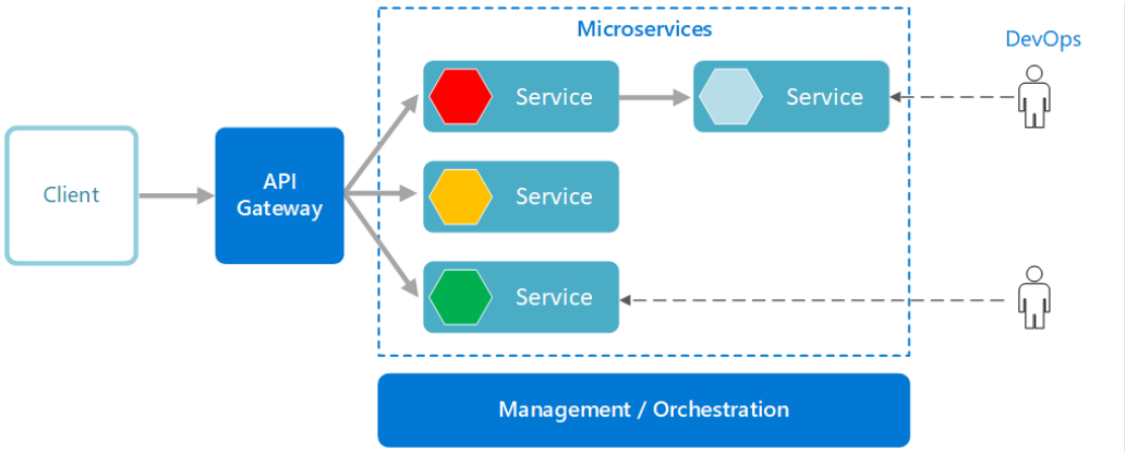



**Figura 2.2:** schema microservizio[2]

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:

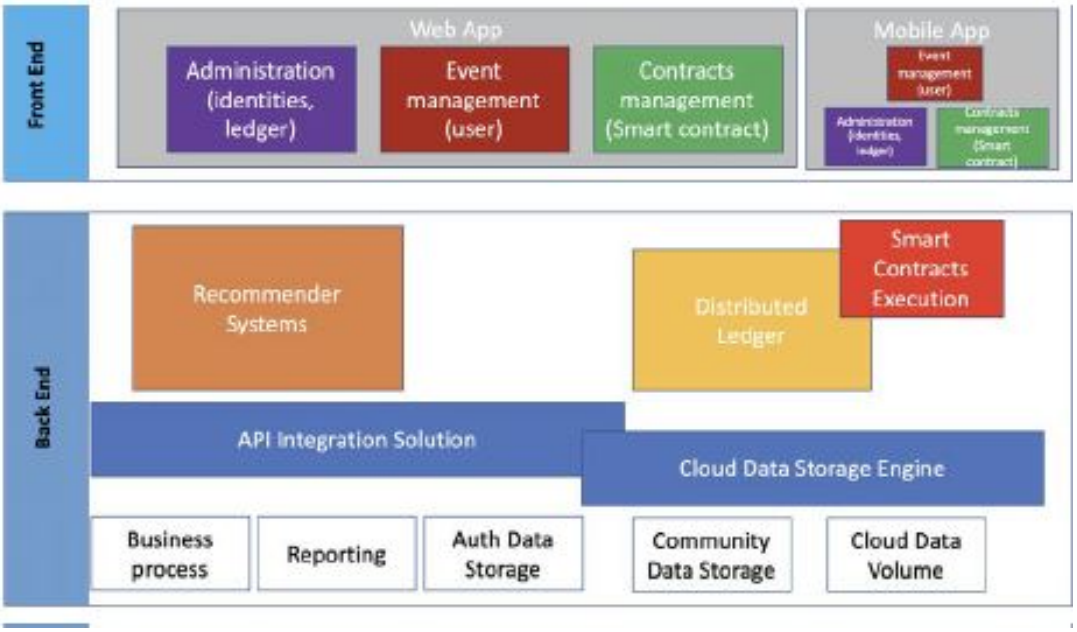



**Figura 3.1:** Schema a blocchi della piattaforma DME

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 micro servizi;

- 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 obietivi:

- 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 market place di definizione di beni e servizi, definizione di utenti e profili, servizi orizzontali di autorizzazione e autenticazione, back up 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:**

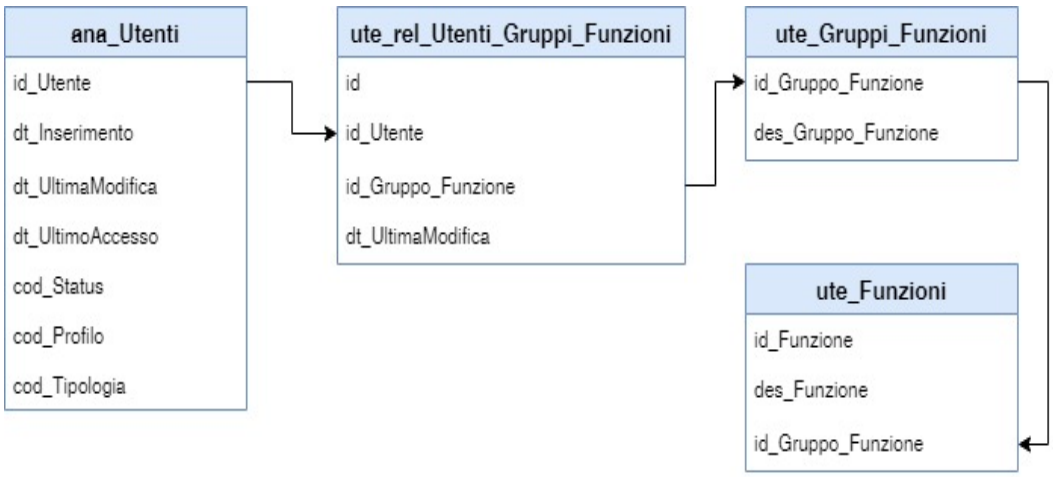



**Figura 3.3:** Microservizio Utente

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:**

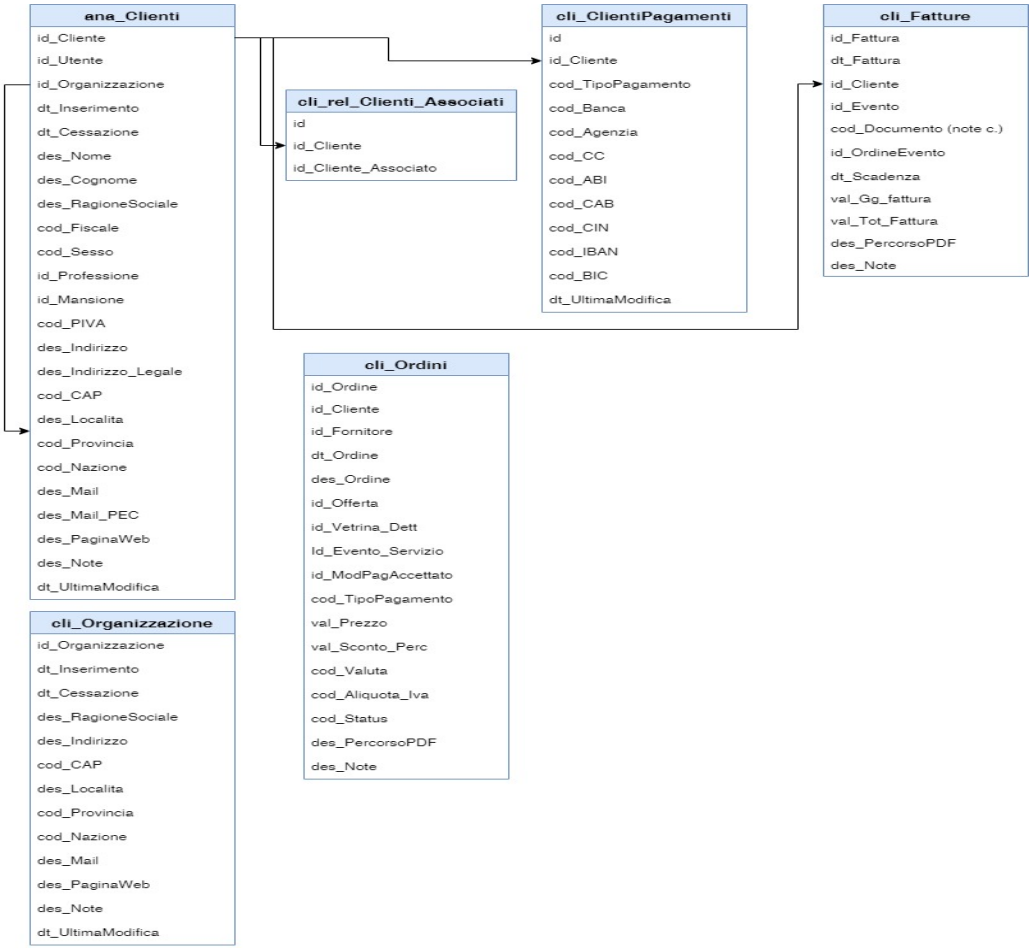



**Figura 3.4:** Microservizio Cliente

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:**

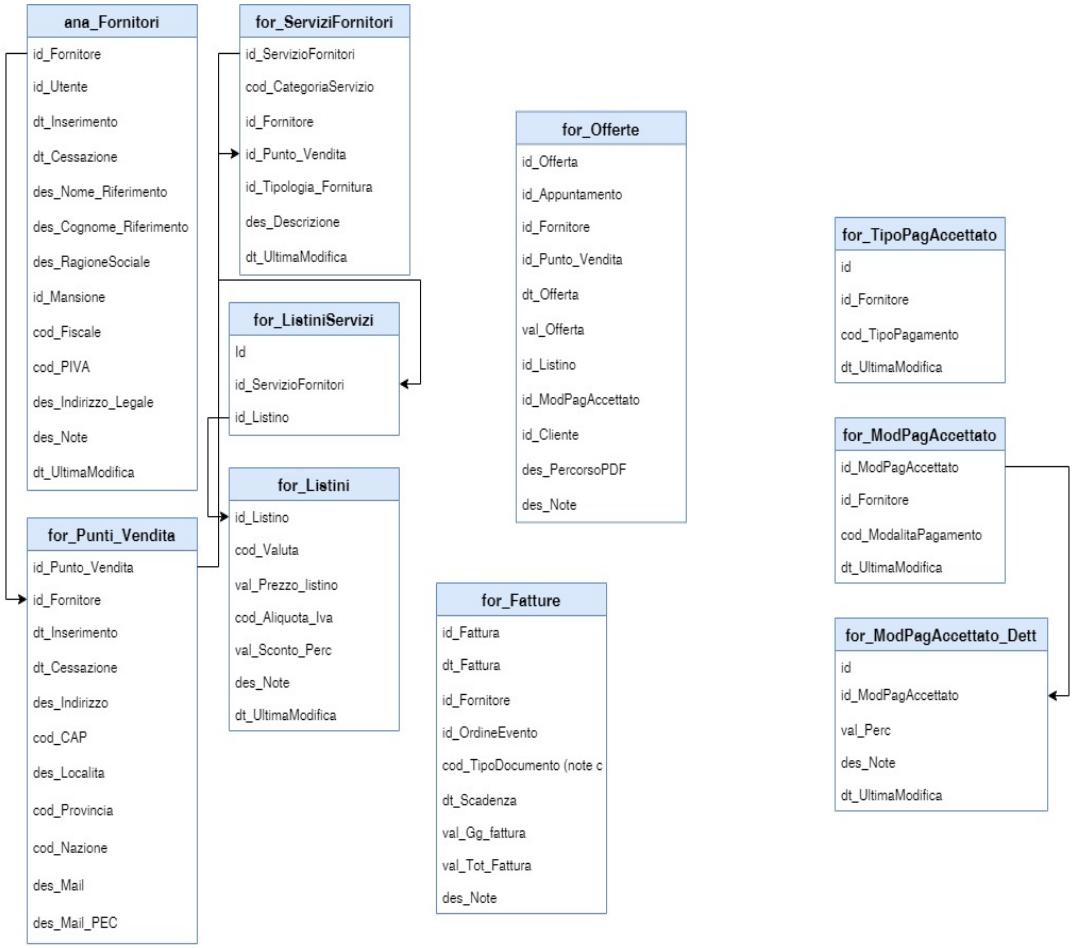
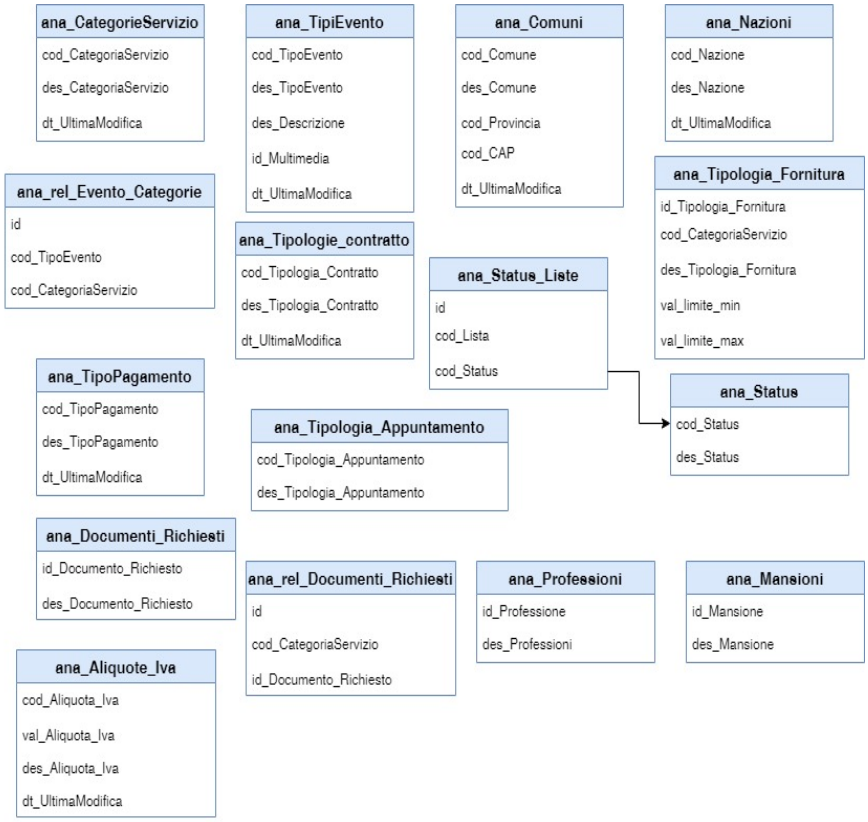



**Figura 3.5:** Microservizio Fornitore

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:**

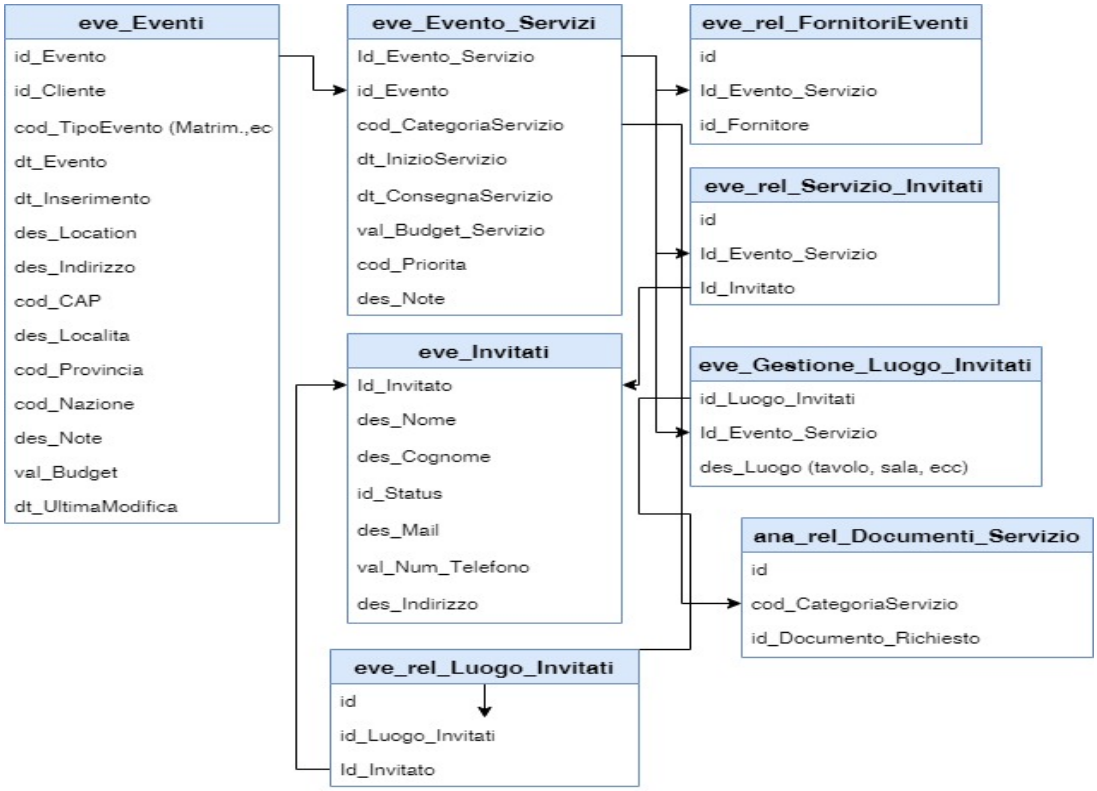



**Figura 3.7:** Microservizio Eventi

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:**

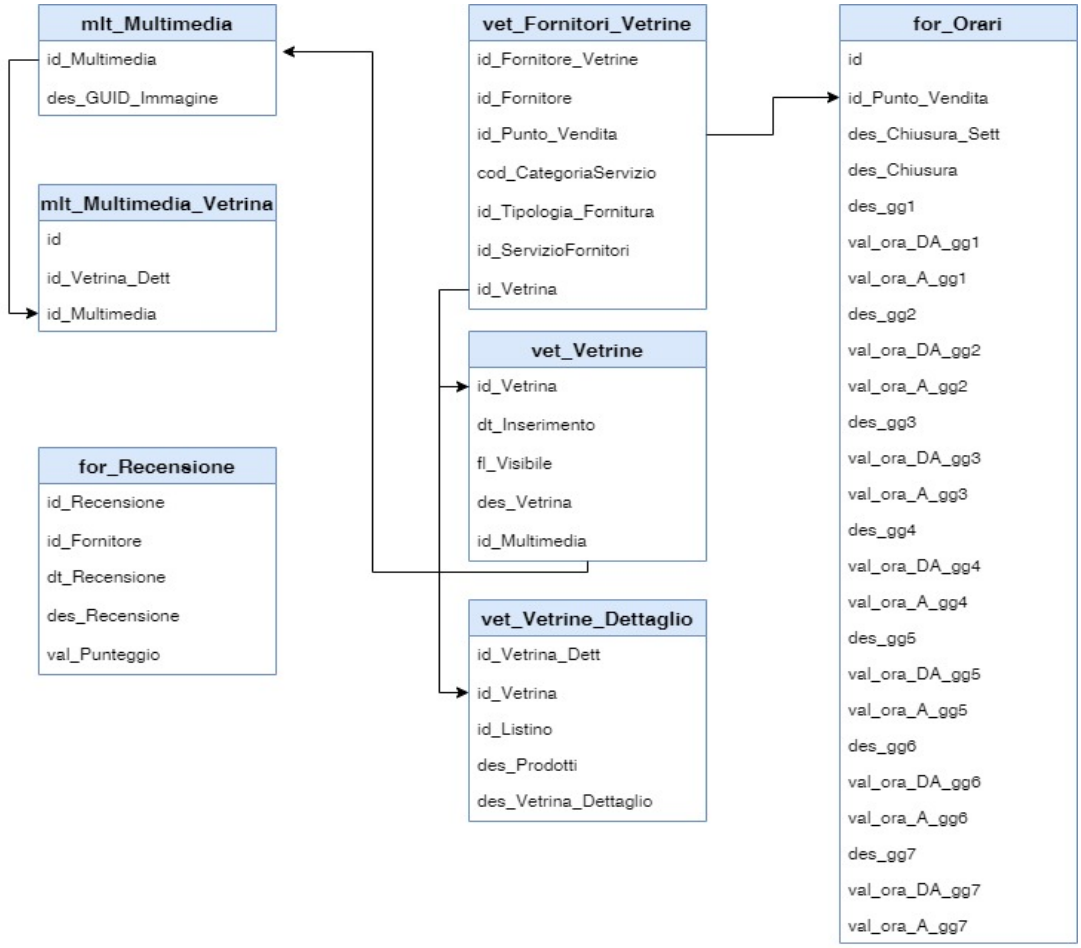



**Figura 3.9:** Microservizio Vetrine

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:**

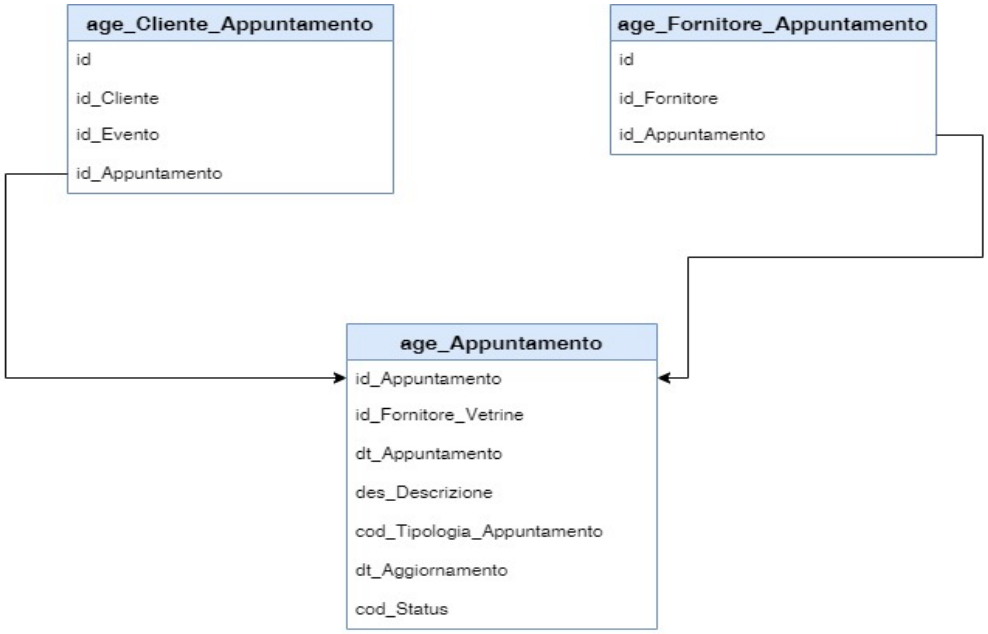



**Figura 3.10:** Microservizio Agenda

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.

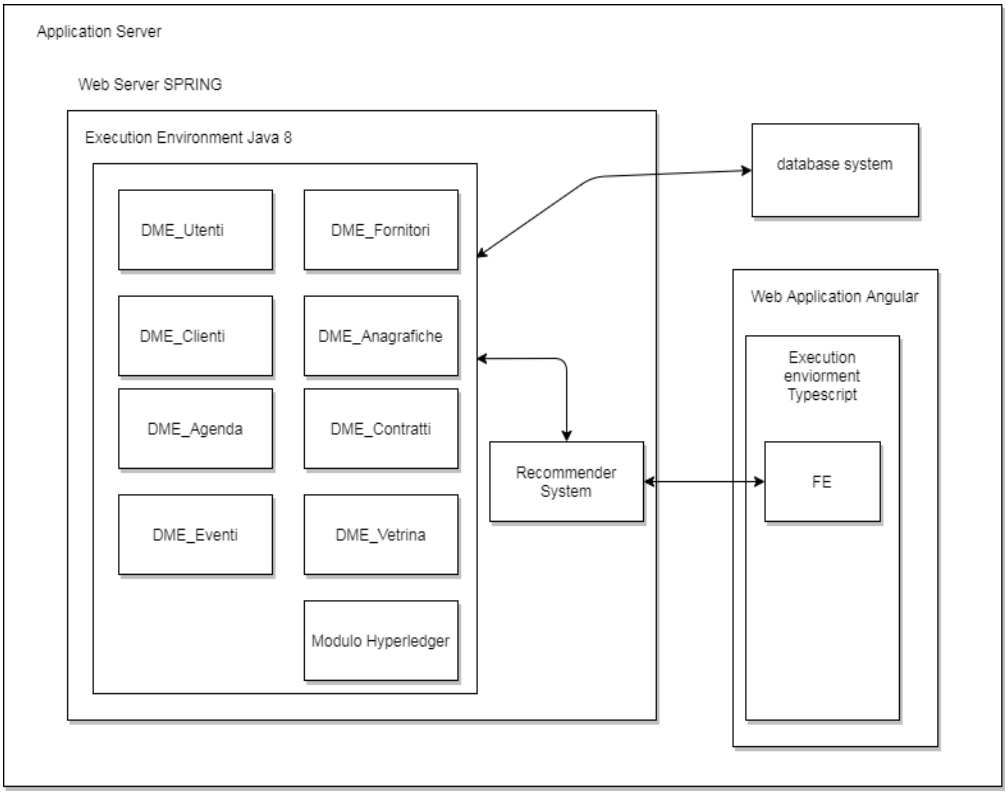



**Figura 3.11:** Deployment diagram

## 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:

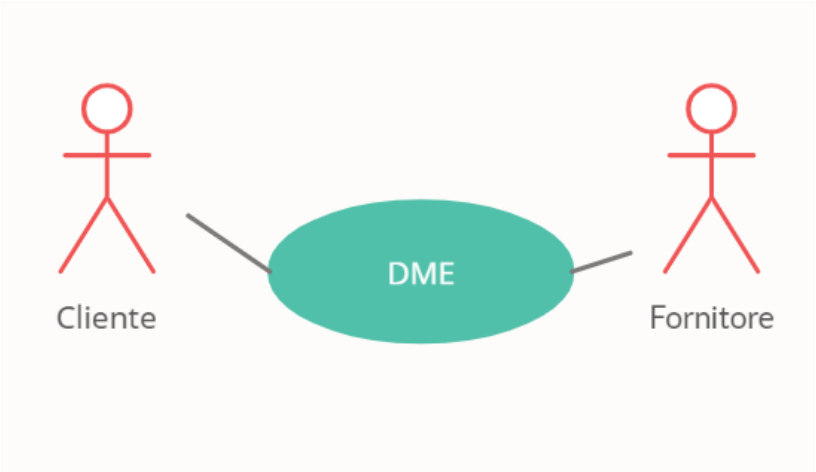



**Figura 4.1:** Context Diagram

### 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;<br>2. Il sistema mostra la pagina del form;<br>3. L'utente compila il form con i suoi dati e email corretta;<br>4. Il sistema verifica il contenuto e che la mail sia valida, in caso positivo manda mail di verifica;<br>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;<br>2. Il sistema mostra gli eventi possibili da creare e il costo relativo al servizio;<br>3. L'utente seleziona "Matrimonio";<br>4. Il sistema mostra le tipologie di pagamento;<br>5. L'utente ne seleziona una tipologia;<br>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);<br>3.a.1 Il sistema verifica che l'account è valido;<br>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;<br>2. Il sistema mostra la lista della tipologia dei vari fornitori selezionabili e un campo per immettere la stima di quanto si vorrà spendere;<br>3. L'utente seleziona i vari campi e immette il prezzo stimato;<br>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;<br>2. Il sistema mostra la percentuale delle attività completate e in ordine di priorità le attività da svolgere.;<br>3. L'utente seleziona l'attività da svolgere;<br>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;<br>2. Il sistema mostra le chiese disponibili per quella data;<br>3. L'utente clicca sul tasto ordina per prenotare la chiesa;<br>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;<br>2. L'utente visiona la lista e seleziona i luoghi desiderati per ricevere un preventivo;<br>3. Il sistema appena riceve i preventivi dai ricevimenti notifica l'utente;<br>4. L'utente prenota e procede al pagamento dell'acconto;<br>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;<br>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;<br>2. Il sistema mostra la lista dei contratti stipulati e se sono conclusi o in sospeso;<br>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;<br>2. Il sistema mostra la lista degli invitati ;<br>3. L'utente può selezionare un invitato e selezionare il suo stato(confermato, rifiuto);<br>4. L'utente può inserire una piantina per la gestione dei tavoli ;<br>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 non 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

# Layer Back-end

## 5.1 Stato dell'arte e strumenti utilizzati

Di seguito verranno descritti i framework utilizzati

### 5.1.1 Maven

Maven, prodotto della Apache Software Foundation, è uno strumento di build automation utilizzato prevalentemente nella gestione di progetti Java. Simile per certi versi a strumenti precedenti, come ad esempio Apache Ant, si differenzia però da quest'ultimo per quanto concerne la compilazione del codice. Con questo strumento infatti non è più necessaria la compilazione totale del codice, ma viene fatto uso di una struttura di progetto standardizzata su template definita archetype. Il vantaggio derivante dall'utilizzo di questo tool è da subito evidente: se generalmente per sviluppare un software sono necessarie numerose fasi, con la build automation l'intero processo viene automatizzato, riducendo il carico di lavoro del programmatore e diminuendo le possibilità di errore da parte dello stesso. Alcune delle fasi fondamentali che vengono automatizzate dal tool sono la compilazione in codice binario, il packaging dei binari, l'esecuzione di test per garantire il funzionamento del software, il deployment sui sistemi ed infine la documentazione relativa al progetto portato a termine. Esistono poi alcuni tratti comuni ai prodotti di build automation, a partire dal build file, che contiene tutte le operazioni svolte. Per capire meglio che cos'è Maven e come funziona, dopo aver evidenziato i progetti automatizzati, passiamo a considerare quelli che sono i suoi componenti principali e a spiegarne l'utilità ai fini della realizzazione di un progetto software. La prima caratteristica dello strumento è la presenza del file pom.xml, acronimo di Project Object Model, che ha il compito di definire identità e struttura del progetto. Il file pom.xml è diviso a sua volta in cinque parti, la relazione tra diversi file pom.xml, le build settings, la project information, il build environment e la configurazione

dell'ambiente Maven. Un'altra componente importante di Maven è rappresentata dai goal, ovvero l'insieme delle funzioni che possono essere eseguite sui diversi progetti. Poi bisogna menzionare la cartella repository, tramite la quale l'utente è in grado di gestire il sistema delle librerie. Ultima componente fondamentale del tool è il file settings.xml, usato per la configurazione di repository, proxy e profili. La particolarità di questa componente è che può essere specificata su due livelli, sotto user.home per il singolo utente oppure sotto maven.home per più utenti.

### File Pom

POM sta per Project Object Model e ogni progetto è descritto attraverso il file di configurazione pom.xml. Il POM è un file XML e guida l'esecuzione in Maven definendo in modo chiaro l'identità e la struttura di un progetto in ogni suo aspetto. Tutto è descritto nel POM: informazioni generali del progetto, dipendenze, processo di compilazione e fasi secondarie come la generazione di documentazione. Un POM ai minimi termini è composto da un tag root project che conterrà i tag:

- modelVersion che dichiara a quale versione di POM questo progetto è conforme (a noi ci basta sapere che deve essere settato a 4.0.0)
- groupId ID del gruppo del progetto
- artifactId ID dell'artefatto (del progetto)
- version cioè la versione del progetto
- packaging che è il tipo di archivio che vogliamo esportare (jar, war o ear, come vedremo troveremo in una cartella chiamata "target" l'archivio esportato)

I parametri groupId, artifactId, version e packaging identificheranno univocamente un progetto. Se packaging non è specificato nel POM, assumerà "jar" a valore di default. Per ogni dipendenza è possibile anche definire uno scope:

- compile (default) – le dipendenze sono disponibili in tutti i classpath del progetto
- provided – è simile a compile, ma prevede che a runtime le dipendenze siano rese disponibili dall'ambiente di esecuzione (per esempio le JavaEE APIs per un'applicazione enterprise)
- runtime – le dipendenze sono richieste solo in esecuzione
- test – le dipendenze sono richieste solo per la compilazione e l'esecuzione dei test

- system – la dipendenza non viene recuperata tramite repository, ma ne viene esplicitamente dichiarata la posizione locale

Maven permette anche la possibilità di definire dei goal. Un goal è una singola funzione che può essere eseguita sul progetto. I Goal possono essere sia specifici per il progetto dove sono inclusi, sia riusabili. I Plugin sono goal riutilizzabili in tutti i progetti. Maven ha una serie di plugin built-in disponibili, i principali sono i seguenti:

- compile (default) – le dipendenze sono disponibili in tutti i classpath del progetto
- compiler: che permette di compilare i file sorgenti
- deploy: che permette di depositare il pacchetto generato nel repository remoto
- install: che permette di depositare il pacchetto generato nel repository locale
- site: che permette di generare la documentazione del progetto
- archetype: che permette di generare la struttura di un progetto a partire da un template

Ciascun plugin mette a disposizione dei goal specifici o più di un goal. Ciascun goal, a sua volta, riceve in ingresso specifici parametri, facoltativi o obbligatori. Altri plugin possono essere realizzati qualora sia necessario estendere ulteriormente le capacità di Maven a causa di particolari esigenze.[3]

### 5.1.2 Tomcat

Tomcat è un Servlet Engine (chiamato anche Servlet Container o Servlet/JSP); è cioè un software capace di gestire Web applications scritte secondo le specifiche da utilizzare per il linguaggio Java della Sun Microsystem. Tomcat è un programma open source della Apache Software Foundation nato in seno al cosiddetto Progetto Jakarta, possiede molti tratti in comune con Apache. In effetti questo Servlet Engine è in tutto è per tutto un Web server in quanto interpreta le richieste provenienti dai browser dei client e provvede a generare dinamicamente i relativi output. Rispetto ad Apache, Tomcat ha però delle caratteristiche particolari che descriveremo in questo capitolo. Digitando le URL, visualizziamo delle determinate aree controllate da un Web server, ogni area in Tomcat prende il nome di Contesto e ogni Contesto fa capo oggetti definiti validi sia per il loro ambito di azione che per gli utenti che ad esso accedono. I Contesti non vengono generati sulla base di chiamate, ma dal momento in cui viene avviato il processo di Tomcat e come quest'ultimo attendono di rispondere alle richieste degli utenti senza mai arrestarsi. Ad ogni utente viene

affidato dal Web server un Contenitore personale detto Sessione caratterizzato dalle risorse a cui quel determinato utente ha la possibilità di accedere. All'interno di questa sessione e per tutta la durata del suo collegamento alle risorse gestite da Tomcat, l'utente potrà effettuare delle richieste sotto forma di input da soddisfare tramite risposte generate dinamicamente in output. Le richieste a loro modo sono dei Contenitori particolari, in quanto esistono solo nel lasso di tempo che separa l'invio della domanda e la fornitura della relativa risposta. Chi conosce bene i meccanismi che regolano i Web server non avrà di certo notato consistenti novità nel criterio di gestione input/output appena descritto, per quanto riguarda i metodi possiamo invece osservare sostanziali particolarità: all'atto della richiesta, classicamente la digitazione di una URL tramite protocollo di comunicazione HTTP, viene lanciatal'istanza di un oggetto chiamato Request o Req, questo input porta il Servlet Engine ad istanziare un oggetto chiamato Response o Res come reazione di default alla richiesta del client. Req e Res dovranno quindi essere dirottati verso il Contesto contenente le risorse utili a concludere la transizione in atto. Oltre a Req e Res, Tomcat provvederà ad istanziare un ulteriore oggetto, Ses, riferito al Contenitore di Sessione, con il compito di identificare univocamente il richiedente e di controllare che la sua Sessione sia aperta. Ora entrano in gioco le Servlet a cui abbiamo fatto precedentemente riferimento nell'Introduzione, esse dovranno rendere consistente Res. In pratica il Contesto a cui appartengono le risorse necessarie per la soddisfazione della richiesta dovrà indirizzare queste ultime alla Web application Java necessaria al completamento della transizione. Nel momento in cui verrà generato l'output avremo tre esiti possibili:

- La Servlet esaudisce la richiesta e lascia a Tomcat il compito di consegnare l'output al client (per esempio, sotto forma di ipertesto HTML)
- La risorsa necessaria alla soddisfazione della richiesta non è presente nel Contesto utilizzato, dovrà quindi essere ricercata in un ulteriore Contesto esterno, questo compito verrà quindi affidato a Tomcat che riavvierà la procedura di input/output
- La richiesta può essere soddisfatta all'interno del Contesto ma non tramite la risorsa richiesta, dovrà quindi essere generato un Forward per il reindirizzamento verso la risorsa adeguata.

Tomcat versione 4.x è stato distribuito con tre componenti principali:

- Catalina è il contenitore di servlet Java di Tomcat. Catalina implementa le specifiche di Sun Microsystems per le servlets Java e le JavaServer Pages (JSP, Pagine JavaServer). In Tomcat un elemento del Realm rappresenta un database di username, password e ruoli (analoghi dei gruppi di UNIX) assegnati a quegli utenti. Differenti implementazioni del Realm permettono a Catalina

di essere integrato in ambienti dove tali informazioni di autenticazione sono già state create e supportate, e poi gli permettono di utilizzare tali informazioni per implementare una cosiddetta "Container Managed Security" come descritto nelle Specifiche delle Servlet

- Coyote è il componente "connettore HTTP" di Tomcat. Supporta il protocollo HTTP 1.1 per il web server o per il contenitore di applicazioni. Coyote ascolta le connessioni in entrata su una specifica porta TCP sul server e inoltra la richiesta al Tomcat Engine per processare la richiesta e restituire una risposta al client richiedente

- Jasper è il motore JSP di Tomcat. Tomcat 8.x utilizza Jasper 2, che è un'implementazione delle specifiche 2.3 delle Pagine JavaServer (JSP). Jasper analizza i file JSP per compilarli in codice Java come servlet (che verranno poi gestite da Catalina). Al momento di essere lanciato, Jasper cerca eventuali cambiamenti avvenuti ai file JSP e, se necessario, li ricompila. [4]

### 5.1.3 Spring

Nato come codice allegato al libro "Expert One-on-One J2EE Design and Development", Spring ha il chiaro intento di gestire la complessità legata allo sviluppo di applicazioni enterprise. La sua prima apparizione risale al 2003 sotto licenza Apache, ma fu solo nel marzo del 2004 il primo importante rilascio, ovvero la versione 1.0; attualmente è giunto alla versione 5.0.2 e viene definito come il più popolare framework per lo sviluppo di applicazioni Java Enterprise. Visto come una valida, e ormai ben solida alternativa al modello basato su Enterprise Javabeans, questo framework supera l'appesantimento derivato dall'utilizzo forzato di interfacce EJB di tipo home e remote, troppo invasive nel codice scritto, pur mantenendo la possibilità di gestire il descrittore di deployment in XML grazie a nuovi ed innovativi modelli di programmazione, come l'Aspect Oriented Programming (AOP) e l'Inversion of Control (IoC). Spring è inoltre un framework leggero: grazie alla sua struttura estremamente modulare è possibile utilizzarlo nella sua interezza o solo in parte, senza stravolgere l'architettura del progetto; questa peculiarità permette una facile integrazione anche con altri framework già esistenti, come Struts3 o Hibernate.4 Fornisce, inoltre, una serie completa di strumenti per gestire la complessità dello sviluppo software, fornendo un approccio semplificato sia più comuni problemi di sviluppo (accesso ai database, gestione delle dipendenze, etc.) che di testing.
Il framework Spring ha una struttura modulare, il modulo core che fornisce le funzionalità fondamentali del framework. Tale modulo si basa sulle seguenti funzionalità e pattern: Dependency Injection, Aspect-Oriented Programming, la gestione delle transazioni, la struttura di applicazioni Web, l'accesso ai dati, la messaggistica,

i test e altro. Spring permette la progettazione e sviluppo di applicazioni web grazie al suo modulo Spring MVC il quale mette a disposizione le funzionalità core di Spring, accennate al paragrafo precedente, su un pattern architetturale MVC.

### Il pattern MVC

Spesso, quando si implementa un'applicazione Web utilizzando la tecnologia Java, a causa dell'utilizzo di un protocollo come HTTP si tende ad usare in modo indiscriminato Servlet1 e pagine JSP. Se questo da un lato dà la possibilità di una maggiore flessibilità e libertà di programmazione, dall'altro può andare contro alle norme della qualità del software, come riusabilità, portabilità e manutenibilità.
Per sopperire a tali esigenze, si è designata un'architettura per le Web application chiamata Model-View-Controller

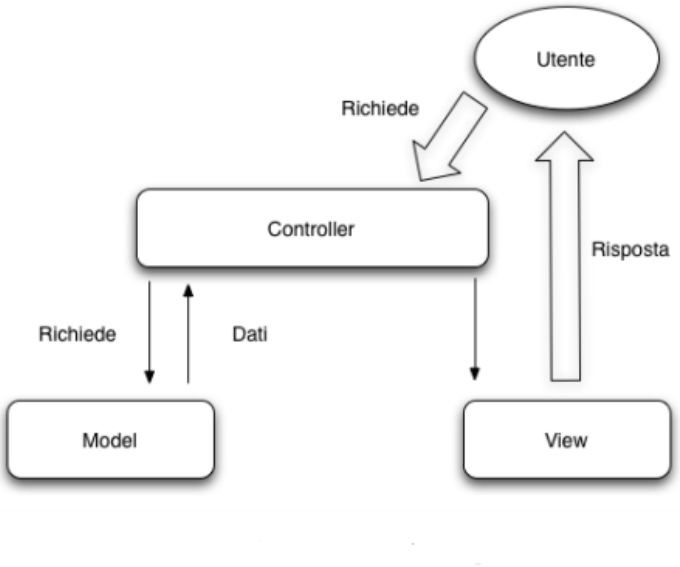



**Figura 5.1:** Schema pattern MVC

Tale architettura è stata introdotta per la prima volta da Trygve Reenskaug nel tardo 1970, ma diventerà una pietra miliare per l'architettura di tutte le future applicazioni Web-based, poichè riesce brillantemente a separare tutta la logica business e la rappresentazione delle informazioni dalle interazioni utente con quest'ultime.

Tale modello è costituito sostanzialmente da 3 strati: la parte business formata da Model e Controller, e l'interfaccia utente (View). Nel dettaglio:

- Model: Implementa la logica di business, fornendo i metodi utili per l'accesso ai dati dell'applicazioneed ha la responsabilità della gestione del database
- Controller: Implementa la logica di controllo, riceve i comandi dell'utente (in genere attraverso lo strato View) e li attua modificando lo stato degli altri due componenti
- View: Implementa la logica di presentazione, interpretando i risultati ottenuti da una dal Model e gestendo l'interazione con gli utenti

È importante far notare che sia lo strato View che lo strato Controller dipendono direttamente dal Model, il quale non dipende dagli altri. Questo è uno dei fattori più importanti di questa architettura, poiché permette al modello di essere implementato e testato indipendentemente dallo strato di visualizzazione. Grazie a questi accorgimenti, molteplici sono i benefici derivati: grazie a tale approccio, è possibile implementare viste multiple, permettendo l'esposizione degli stessi dati allo stesso instante in modi diversi. Si riduce la complessità di sviluppo, e conseguentemente il costo di aggiornamento, permettendo la manutenzione ad uno degli strati senza coinvolgerne altri.

### Inversion of Control (IoC)

L'Inversion of Control, già citata in precedenza, è un principio architetturale nato alla fine degli anni ottanta, basato sul concetto di invertire il controllo del flusso di sistema (Control Flow) rispetto alla programmazione tradizionale. Nella Programmazione tradizionale lo sviluppatore definisce la logica del flusso di controllo, specificando le operazioni di creazione, inizializzazione degli oggetti ed invocazione dei metodi. Nell' Inversion of Control (IOC) si inverte il control flow, facendo in modo che non sia più lo sviluppatore a doversi preoccupare di questi aspetti, ma il framework, che reagendo a qualche stimolo se ne occuperà per suo conto.
Una specifica implementazione dell'IoC è la Dependency Injection (DI). La DI prevede che tutti gli oggetti all'interno della nostra applicazione accettino le dipendenze, ovvero gli altri oggetti di cui hanno bisogno, tramite costruttore o metodi setter. Non sono quindi gli stessi oggetti a creare le proprie dipendenze, ma esse vengono iniettate dall'esterno.
L'ecosistema all'interno del quale le applicazioni Spring vivono viene definito IoC container. Lo IoC container si occupa di istanziare gli oggetti (beans) dichiarati nel progetto e di reperire e iniettare tutte le dipendenze ad essi associate. Tali dipendenze possono essere componenti del framework o altri bean dichiarati nel contesto applicativo. [5]

### 5.1.4 Hibernate

In informatica Hibernate è una piattaforma middleware open source per lo sviluppo di applicazioni Java, attraverso l'appoggio al relativo framework, che fornisce un servizio di Object-relational mapping (ORM) ovvero gestisce la persistenza dei dati sul database attraverso la rappresentazione e il mantenimento su database relazionale di un sistema di oggetti Java. Come tale dunque, nell'ambito dello sviluppo di applicazioni web, tale strato software si frappone tra il livello logico di business o di elaborazione e quello di persistenza dei dati sul database (Data Access Layer). È stato originariamente sviluppato da un team internazionale di programmatori volontari coordinati da Gavin King; in seguito il progetto è stato proseguito sotto l'egida di JBoss, che ne ha curato la standardizzazione rispetto alle specifiche Java EE. Hibernate è distribuito in licenza LGPL sotto forma di librerie software da linkare nel progetto di sviluppo software. Lo scopo principale di Hibernate è quello di fornire un mapping delle classi Java in tabelle di un database relazionale; sulla base di questo mapping Hibernate gestisce il salvataggio degli oggetti di tali classi su database (tipicamente attributi di oggetti per ciascun campo dati della tabella). Si occupa inoltre al rovescio del reperimento degli oggetti dal database, producendo ed eseguendo automaticamente le query SQL necessarie al recupero delle informazioni e la successiva reistanziazione dell'oggetto precedentemente "ibernato" (mappato su database). L'obiettivo di Hibernate è quello di esonerare lo sviluppatore dall'intero lavoro relativo alla persistenza dei dati. Hibernate si adatta al processo di sviluppo del programmatore, sia se si parte da zero sia se da un database già esistente. Hibernate genera le chiamate SQL e solleva lo sviluppatore dal lavoro di recupero manuale dei dati e dalla loro conversione, mantenendo l'applicazione portabile in tutti i database SQL. Hibernate fornisce una persistenza trasparente per Plain Old Java Object (POJO); l'unica grossa richiesta per la persistenza di una classe è la presenza di un costruttore senza argomenti. In alcuni casi si richiede un'attenzione speciale per i metodi equals() e hashcode(). Hibernate è tipicamente usato sia in applicazioni Swing che Java EE facenti uso di servlet o EJB di tipo session beans. La versione 3 di Hibernate arricchisce la piattaforma con nuove caratteristiche come una nuova architettura Interceptor/Callback, filtri definiti dall'utente, e annotazione stile JDK 5.0 (Java's metadata feature). Hibernate 3 è vicino anche alle specifiche di EJB 3.0 (nonostante sia stato terminato prima di EJB 3.0 le specifiche erano già state pubblicate dalla Java Community Process) ed è usato come spina dorsale per l'implementazione EJB 3.0 di JBoss. [6]

## 5.2 Implementazione

Il layer Back-end è un server REST strutturato a microservizi. L'architettura a microservizi è già stata descritta precedentemente. Di seguito verrà descritta la filosofia REST

### 5.2.1 REST

Representational state transfer (REST) è uno stile architetturale (di architettura software) per i sistemi distribuiti. L'espressione representational state transfer e il suo acronimo, REST, furono introdotti nel 2000 nella tesi di dottorato di Roy Fielding, uno dei principali autori delle specifiche dell'Hypertext Transfer Protocol (HTTP), e vennero rapidamente adottati dalla comunità di sviluppatori Internet. Il termine REST rappresenta un sistema di trasmissione di dati su HTTP senza ulteriori livelli, quali ad esempio SOAP. I sistemi REST non prevedono il concetto di sessione, ovvero sono stateless, come approfondito successivamente. L'architettura REST si basa su HTTP. Il funzionamento prevede una struttura degli URL ben definita che identifica univocamente una risorsa o un insieme di risorse e l'utilizzo dei metodi HTTP specifici per il recupero di informazioni (GET), per la modifica (POST, PUT, PATCH, DELETE) e per altri scopi (OPTIONS, ecc.).

#### Vincoli

L'approccio architetturale REST è definito dai seguenti sei vincoli applicati ad una architettura, mentre lascia libera l'implementazione dei singoli componenti.

- Client–server. Un insieme di interfacce uniformi separa i client dai server. Questa separazione di ruoli e compiti significa che, per esempio, il client non si deve preoccupare del salvataggio delle informazioni, che rimangono all'interno dei singoli server. In questo modo la portabilità del codice del client ne trae vantaggio. I server non si devono fare carico dell'interfaccia grafica o dello stato dell'utente, in questo modo l'hardware può essere più semplice e maggiormente scalabile. Server e client possono essere sostituiti e sviluppati indipendentemente fintanto che l'interfaccia non viene modificata

- Stateless. La comunicazione client–server è vincolata in modo che nessun contesto client venga memorizzato sul server tra le richieste. Ciascuna richiesta dai vari client contiene tutte le informazioni necessarie per richiedere il servizio e lo stato della sessione è contenuto nel client. Lo stato della sessione può anche essere trasferito al server attraverso un altro servizio di memorizzazione persistente, per esempio un database

- Cacheable. Come nel World Wide Web, i client possono mettere in cache le risposte. Queste devono in ogni modo definirsi implicitamente o esplicitamente cacheable o no, in modo da prevenire che i client possano riutilizzare stati vecchi e dati errati. Una corretta gestione della cache può ridurre o parzialmente eliminare le comunicazioni client-server migliorando scalabilità e prestazioni

- Layered system. La struttura del sistema "a strati" (letteralmente dall'inglese) rende possibile per esempio pubblicare le API in un server, memorizzare i dati in un secondo server e gestire l'autenticazione delle richieste in un terzo server. Questo comporta che un client non può sapere se è connesso direttamente a un server finale oppure a uno della catena

- Code on demand. (opzionale) I server possono temporaneamente estendere o personalizzare le funzionalità del client trasferendo del codice eseguibile. Per esempio questo può includere componenti compilati come Applet Java o linguaggi di scripting lato client come per esempio JavaScript. Il concetto di Code on demand è l'unico vincolo opzionale per la definizione di un'architettura REST

- Uniform interface. Un'interfaccia di comunicazione omogenea tra client e server permette di semplificare e disaccoppiare l'architettura, per poterla modificare separatamente a blocchi

### Risorse

Un concetto importante in REST è l'esistenza di risorse (fonti di informazioni), a cui si può accedere tramite un identificatore globale (un URI). Per utilizzare le risorse, le componenti di una rete (componenti client e server) comunicano attraverso una interfaccia standard (per esempio HTTP) per scambiare rappresentazioni di queste risorse, ovvero il documento che trasmette le informazioni. Per esempio, una risorsa cerchio potrebbe accettare e restituire una rappresentazione che specifica un punto per il centro e il raggio in formato SVG, ma potrebbe anche accettare e restituire una rappresentazione che specifica tre punti distinti qualsiasi lungo la circonferenza nel formato CSV. Un numero qualsiasi di connettori (client, server, cache, tunnel ecc.) può mediare la richiesta, ma ogni connettore interviene senza conoscere la "storia passata" delle altre richieste; proprio per questo motivo l'architettura REST si definisce stateless, in opposizione ad altre architetture o protocolli stateful. Di conseguenza un'applicazione può interagire con una risorsa conoscendo due cose: l'identificatore della risorsa e l'azione richiesta. Non serve sapere se ci sono proxy, gateway, firewall, tunnel o altri meccanismi intermedi tra essa e il server in cui è presente l'informazione necessaria. L'applicazione comunque deve conoscere il formato dell'informazione restituita, ovvero la sua rappresentazione. Tipicamente

è un documento HTML, XML o JSON, ma possono essere anche immagini o altri contenuti.
Di seguito è descritto come i metodi HTTP sono tipicamente usati in una API RESTful.

Se l'url indica una collezione (http://abc.com/resources)

- GET restituisce tutti gli elementi della collezione
- PUT Sostituisce l'intera collezione con un'altra collezione
- POST Crea un nuovo elemento nella collezione. Il codice di stato solitamente restituito è il 201 (Created). L'URI della nuova risorsa viene assegnato automaticamente ed è usualmente restituito da questa operazione (header Location)
- DELETE Elimina l'intera collezione

Se l'url indica un elemento (http://abc.com/resources/item8)

- GET Recupera una rappresentazione dell'elemento indirizzato nella collezione (identificato da item8), in un formato di dato (media type) appropriato
- PUT Sostituisce l'elemento indirizzato nella collezione, o se non esiste, lo crea
- POST Tratta l'elemento della collezione secondo i propri diritti e crea un nuovo elemento all'interno
- DELETE Elimina l'elemento identificato nella collezione

Il metodo GET è un metodo sicuro, ovvero un safe method o nullipotente, il che significa che la sua invocazione non produce alcun effetto collaterale: recuperare o accedere un record non lo modifica. I metodi PUT e DELETE sono idempotenti, ossia lo stato del sistema rimane invariato indipendentemente dal numero di volte che la stessa richiesta viene ripetuta. A differenza dei web services basati su SOAP, non esiste alcuno standard ufficiale per le API web RESTful. Questo perché REST è un insieme di linee guida per un'architettura, mentre SOAP è un protocollo.

### 5.2.2 Struttura del singolo microservizio

L'obiettivo del singolo microservizio è permettere al layer di front-end di ricevere i dati delle tabelle del DB definendo delle API secondo la filosofia REST. Per dialogare con il DB avremo bisogno di definire delle entità che rappresentano gli oggetti di dominio della nostra applicazione, le entity. Una volta che la struttura della tabella è stata decisa e creata sul DB possiamo iniziare a scrivere le nostre entity che rispecchino lo schema della tabella.

### Entity

Una entity è una classe annotata con @Entity della libreria javax.persistence che indica le classi-entità e con @Table che specifica il nome della tabella e lo schema del DB che la classe rappresenta. Ogni attributo della classe rappresenta una colonna della tabella e deve essere annotato con @Column per specificarne il nome se questo non corrisponde. La chiave primaria della tabella deve essere annotata con @Id e con @GeneratedValue per poter generare automaticamente un nuovo id sequenziale delegando questo compito al DB. Per evitare di scrivere costruttori, getter e setter si è utilizzato la libreria di Lombok che permette di generare automaticamente tale codice tramite le annotazioni @Data, @AllArgConstructor, @NoArgConstructor e @Builder. Per evitare problemi di inconsistenza di Hibernate si sono riscritti i metodi equals(Object o) ed hashcode(). La specifica JPA prevede inoltre una serie di annotazioni utili per definire le relazioni esistenti tra le varie classi/entità/tabelle. Questo, ad esempio, permette a strumenti come Hibernate di recuperare contestualmente tutti gli oggetti collegati tra loro, in base alle relazioni dichiarate. Nel nostro caso, avendo una struttura a microservizi, si vogliono tenere le tabelle separate e queste annotazioni sono state omesse.

### Repository

Il fulcro di Spring Data è costituito dalle interfacce Repository e JpaRepository. La prima funge da marker, identificando le interfacce delegate alla comunicazione con la sorgente dati, la seconda introduce le principali operazioni CRUD, tipiche di ogni Repository. In Spring Data è possibile estendere il comportamento del JpaRepository in due modi: definendo Query Method o implementando esplicitamente metodi custom. La funzionalità principale introdotta da Spring Data è costituita proprio dai Query Method. Basterà infatti dichiarare dei semplici metodi nella nostra interfaccia-repository per costruire le query di cui abbiamo bisogno. Per far sì che il framework sappia come convertire i nostri metodi in query, la dichiarazione deve rispettare le regole sintattiche descritte nella documentazione.
Ad esempio:
List<NomeEntità> findByClienteId(long id);
Spring Data JPA provederà a convertire (utilizzando JPQL) in una query tipo:
select c from NomeTabellaAssociata c where c.ClienteId = ‘id’
Per ogni classe entity, quindi, è necessario definire un’interfaccia che estenda JpaRepository specificando la classe entity, il tipo usato per la chiave primaria e gli eventuali query method.
Per connettersi al DB è necessaria la scrittura di una classe di configurazione annotata con @Configuration che permette la dichiarazione di metodi annotati con @Bean. E’ stato definito un metodo che crea una connessione SSH con il database

specificando un url, i driver utilizzati dal DB, username e password che sono stati forniti dall'azienda.

#### DTO

L'Oggetto di Trasferimento Dati o Data Transfer Object in sigla DTO è un design pattern usato per trasferire dati tra sottosistemi di un'applicazione software. La differenza tra le entity è che un DTO non ha alcun comportamento se non di archiviare e recuperare i suoi dati. I DTO sono delle semplici classi che servono a mappare i Json che vengono scambiati con le chiamate http tra front-end e back-end. I DTO hanno gli stessi attributi della classe entity associata di cui vogliono trasmettere i dati ma possono anche contenere dati aggiuntivi su eventuali stati dell'applicazione.

#### Controller

I controller sono classi annotate con @Controller e si occupano di mappare gli url che verranno usati dalle chiamate http con dei metodi per la costruzione di una risposta. I controller sono annotati con @RequestMapping che specifica la prima parte dell'url. Ogni metodo è poi annotato con il tipo di richiesta GET, POST, PUT o DELETE. Il Controller ha il compito di ricevere un DTO e mapparlo in una entity e passarlo alla logica dell'applicazione. Ricevuto una entity risultato deve mapparla in un DTO da inserire in una risposta http. In caso di errore ha il compito di costruire una risposta http con il codice di errore adeguato. Il mapping tra entity e DTO viene fatto da una classe chiamata ModelMapper che, se gli attributi delle due classi hanno gli stessi nomi, riesce a fare il mapping in autonomia, altrimenti può ricevere un metodo di mapping custom.

#### Service

Il service è la classe che contiene la logica dell'applicazione e si occupa di far comunicare i controller con il DB attraverso i metodi definiti nelle interfacce repository. La classe è annotata con service e i suoi metodi vengono chiamati dal controller. I compiti principali sono quelli di salvare dei nuovi dati ricevuti da una entity con il metodo save() del rispettivo repository o di prelevare i dati salvandoli in una entity e restituirli al controller.

### 5.2.3 Microservizi Realizzati

Di seguito è riportato l'elenco dei microservizi realizzati ed una loro descrizione. Ogni microservizio è mappato su una porta diversa.

- DME_Anagrafiche: serve a memorizzare gli elenchi di opzioni contenute nei form dell'applicazione come l'elenco delle province, dei comuni, delle categorie di servizio, ecc...
- DME_Utenti: definisce gli attributi dell'utente generico, tra cui username, mail e password, le sue API vengono chiamate dall'Auth Server e non direttamente dall'applicazione
- DME_Clienti: definisce il primo tipo di utenti della piattaforma e gli ordini effettuati per un dato servizio
- DME_Fornitori: definisce il secondo tipo di utenti della piattaforma e ciò che caratterizza i loro servizi tra cui i punti vendita e i listini
- DME_Eventi: definisce i singoli eventi alcune loro caratteristiche come il budget, gli invitati e la gestione tavoli
- DME_Vetrina: definisce i dettagli di un singolo servizio fornito da un fornitore
- DME_Agenda: definisce gli appuntamenti presi da clienti e fornitori e il loro stato
- DME_Contratt: definisce uno specchio degli smart contranct nella blockchain per averne una copia anche sul DB

## 5.3 Autenticazione

Per l'autenticazione degli utenti registrati alla piattaforma DME si è utilizzato il protocollo OAuth2.

### 5.3.1 OAuth2

OAuth 2 è un protocollo standard aperto che consente alle applicazioni di accedere alle risorse protette di un servizio per conto dell'utente. OAuth 2.0 definisce flussi di autorizzazione per applicazioni native, applicazioni Web e dispositivi mobili. Molte aziende offrono endpoint OAuth: Google, Facebook, LinkedIn, GitHub sono solo le più famose, ma il loro numero è in costante crescita. Il flusso effettivo dei messaggi del protocollo può variare in base al tipo di grant utilizzato, di seguito si descriverà il Client Credentials grant utilizzato dalla piattaforma. Il Client Credentials grant viene utilizzato principalmente in applicazioni "intra-server", come servizi batch o applicazioni a riga di comando. In questo caso l'Authorization Server concede l'accesso all'applicazione piuttosto che all'utente. Per utilizzare questo flusso, all'applicazione deve essere stato assegnato un client ID e un client

secret. Questi parametri sono generati dall'Authorization Server e sono necessari per garantire che il client che si connette sia effettivamente chi dice di essere. Il processo di autorizzazione funziona in questo modo:

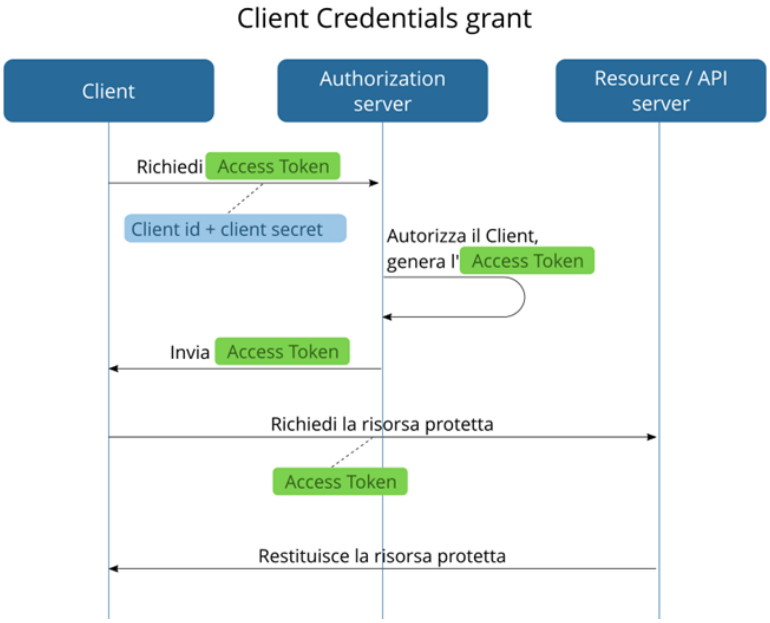



**Figura 5.2:** Protocollo OAuth2

- L'applicazione invia all'Authorization Server una richiesta di autorizzazione assieme ai parametri client id e client secret
- L'Authorization Server verifica le credenziali fornite
- L'Authorization Server restituisce all'applicazione un Access Token ed eventualmente un Referesh Token
- L'applicazione utilizza l'Access Token in ogni successiva richiesta al servizio [7]

### 5.3.2 Implementazione

L'azienda si è occupata dell'implementazione dell'Authorization Server che può essere visto come un ulteriore microservizio che si occupa della registrazione e dell'autenticazione degli utenti. Usando le librerie fornite da Spring Security sono state definite una classe Token e una TokenStore che estendono rispettivamente le

classi JwtToken e JwtTokenStore. Esse si occupano di definire il token scambiato dall'applicazione, quali informazioni contiene e alcune funzionalità per il suo salvataggio.
Per intercettare le chiamate si è creare un filtro custom che estende la classe BasicAuthenticationFilter che decodifica il token presente nelle chiamate http dell'applicazione e, nel caso la decodifica non corrispondesse ad un utente regolare o il token fosse assente, restituisce una risposta di errore. Per l'applicazione del filtro Spring Security fornisce una classe astratta WebSecurityConfigurerAdapter che permette di inserire il nuovo filtro alla filterChain e di specificare anche delle eccezioni per alcuni url.
Infine sono stati definite delle API per la registrazione e l'autenticazione del nuovo utente con la stessa struttura dei microservizi precedenti. [8]

# Capitolo 6

# Conclusioni

## 6.1 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.

## 6.2 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.

## 6.3 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] Codingjam. *Organizziamoci con Maven.* [Online; in data 18-marzo-2021]. 2021. URL: https://codingjam.it/organizziamoci-con-maven-parte-i/ (cit. a p. 53).

[4] Claudio Garau. *Come funziona Tomcat.* [Online; in data 22-marzo-2021]. 2020. URL: https://www.mrw.it/webserver/come-funziona-tomcat_9840.html (cit. a p. 55).

[5] Dario Frongillo. *Spring.* [Online; in data 15-marzo-2021]. 2020. URL: https://italiancoders.it/introduzione-spring-inversion-of-control/ (cit. a p. 57).

[6] Jboss. *Hibernate, Why are HashCode() important?* [Online; in data 22-marzo-2021]. 2018. URL: https://developer.jboss.org/docs/DOC-13933 (cit. a p. 58).

[7] Enrico Trevisan. *OAuth2.* [Online; in data 29-marzo-2021]. 2019. URL: https://www.teranet.it/introduzione-ad-oauth-2/ (cit. a p. 65).

[8] LateraleCloud. *Spring Security.* [Online; in data 17-marzo-2021]. 2020. URL: https://www.lateralecloud.it/sicurezza-api-spring-security-jwt/ (cit. a p. 66).