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

Giulia MELETTI

Aprile 2021

Sommario

Questo progetto di Tesi si è incentrato sullo sviluppo del modulo di Front End della piattaforma Digital Management Events (DME). L'obiettivo della piattaforma è quello di fornire supporto all' organizzazione e gestione degli eventi, occupandosi anche della contrattazione cliente-fornitore attraverso gli smart contract. Per raggiungere questi obiettivi è stato necessario prima di tutto sviluppare l'architettura a microservizi per il lato Back End e successivamente è stato implementato il lato Front End della piattaforma che permette la connessione tra clienti e fornitori. Infine il Front End è stato integrato con Hyperledger per la definizione e gestione degli smart contract.

La progettazione di questo sistema è stata svolta in collaborazione con i colleghi Pietro Cilluffo e Martino Massa. Dopo una fase di analisi e definizione degli obiettivi svolta in team, ognuno dei partecipanti si è occupato di un modulo diverso. Dunque nella trattazione di questo elaborato verranno inizialmente descritte le tecniche di analisi del problema, mirate a produrre le specifiche di sistema. Successivamente verrà esaminata l'implementazione del modulo Front End.

La fase preliminare del progetto è stata l'analisi dei requisiti, ovvero di definizione delle proprietà del prodotto software prima di iniziarne lo sviluppo. Durante questa attività si è proceduto per fasi che hanno portato ad ottenere i requisiti del sistema. Prima di tutto è stata effettuata un'analisi degli stakeholder, dunque delle persone che hanno un interesse nei confronti del sistema. Una volta identificati gli stakeholder è stato definito un context diagram che definisce ciò che è all'interno del sistema da sviluppare e ciò che si trova all'esterno. Conclusa questa fase si è proceduto ad elencare i requisiti funzionali e non funzionali utilizzando due macro metodologie:

- User Centered Design: vengono definite delle story, quindi delle situazioni rappresentative, in cui si potrebbe trovare l'utente e si valuta il tipo di interazione che potrebbe avere con il sistema

- Definizione di scenari e casi d'uso

Terminata la fase di analisi è stata delineata l'architettura della piattaforma (Figura 1) e sono stati definiti i 3 moduli fondamentali con i rispettivi obiettivi:

- Il modulo front-end ha la responsabilità di erogare servizi di accesso e rendere operative sia le funzioni di ricerca dei fornitori, che quelle di gestione del contratto direttamente all' interno della piattaforma. La progettazione prevede semplicità, accesso multi-device e in mobilità.

- Il modulo back-end, è basato su architettura a microservizi in cui ciascun microservizio è composto da funzionalità elementari riusabili. Questo modulo ha come obiettivo la realizzazione di: market place di definizione di beni e servizi, servizi di autorizzazione e autenticazione, back up e recovery.

- Il modulo 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.

Da questo momento in poi verranno approfonditi gli argomenti di progettazione e sviluppo del modulo di Front End.

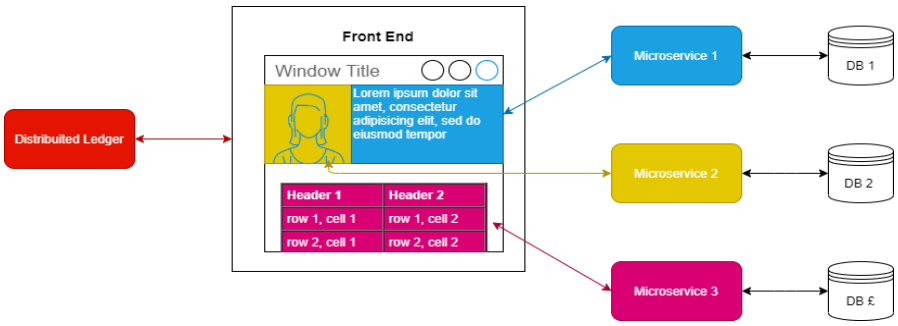


Figura 1: Moduli DME

Il Front End per il progetto Digital Management Events è costituito da una Web Application. In particolare si è scelto di sviluppare una Single Page Application (SPA), dunque un'applicazione che esegue la maggior parte della logica dell'interfaccia utente in un Web browser, comunicando con il server Web principalmente usando le API Web. Per lo sviluppo della SPA è stato scelto il framework Angular. Questa scelta è stata ritenuta la migliore in quanto riesce a soddisfare i requisiti necessari per la piattaforma DME:

- L'applicazione deve esporre un'interfaccia utente avanzata con numerose funzionalità. Le applicazioni a pagina singola supportano funzionalità lato

client complete, che non richiedono di ricaricare la pagina quando gli utenti eseguono operazioni o passano da un'area all'altra dell'app. Inoltre, le SPA vengono caricate rapidamente recuperando i dati in background e facendo risultare più rapide le singole azioni dell'utente in quanto le pagine vengono ricaricate completamente solo in casi sporadici.

- Angular permette, utilizzando lo stile architetturale REST, di comunicare facilmente con il server e ottenere i dati necessari per interagire con le pagine web.

- La Web Application è Cross Platform. Infatti con Angular è possibile creare applicazioni e riutilizzare il codice per qualsiasi target di distribuzione: web, web mobile, mobile nativo e desktop nativo.

Per sfruttare al meglio l'efficienza dell'architettura a microservizi ed avere un'interfaccia utente ottimizzata è stato utilizzato un approccio detto micro-front-end. L'idea di base di un'architettura a micro-front-end consiste nella suddivisione dell'interfaccia complessiva in sotto-funzionalità interattive, il più possibile indipendenti, ognuna vincolata ad un unico microservizio.
Nel progetto DME è stato seguito questo approccio quando le informazioni dei diversi microservizi non necessitavano di essere intrecciate, facendo comunicare i singoli componenti dell'applicazione con i microservizi necessari alla visualizzazione dei dati.

Per adattarsi alla diversità dei dispositivi utilizzati per la navigazione web è stato utilizzato un Design Responsive. Recenti studi industriali dimostrano che la responsività delle applicazioni web influisce direttamente sulla soddisfazione del cliente e, di conseguenza, su metriche business-critical come entrate e tassi di conversione.
Responsive web design significa costruire il layout dell'interfaccia web su griglie fluide in grado di adattarsi dinamicamente a diversi dispositivi di visualizzazione. A livello tecnico, ciò si ottiene utilizzando unità relative (percentuali o em) piuttosto che unità assolute (pixel o punti) per il dimensionamento degli elementi della pagina nonché le mediaquery CSS3 in modo da applicare regole CSS diverse alla posizione e al floating degli elementi. Per soddisfare il requisito di adattabilità dei contenuti in base al dispositivo, nel progetto DME sono stati utilizzati 3 diversi approcci: CSS3 media queries, Bootstrap e alcuni componenti Angular Material.

Mediante la comunicazione tra i moduli di Front End, Back End e Distributed Ledger si è riusciti a soddisfare i requisiti iniziali del progetto ovvero tutte le funzionalità di supporto all'organizzazione e gestione di eventi. Rimangono dei punti aperti sull'effettiva validità legale degli smart contract in quanto per la legislazione attuale non rispetterebbero i vincoli di accordo, causa, oggetto e forma propria di un contratto legale.

Gli sviluppi futuri prevedono l'implementazione di un modulo Recommender System che si occuperà di avvicinare in modo automatico e puntuale la domanda e l'offerta, basandosi su dati come la cronologia dell'utente o le preferenze di persone simili.

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 previsto inizialmente un utilizzo nel mercato del territorio pugliese, con l'obiettivo successivamente di espandersi nel territorio nazionale. La piattaforma digitale è rivolta al supporto di utenti e fornitori che vogliono l'organizzare, gestire, comunicare e partecipare ad eventi di variegate tipologie. Stabilita la vastità del progetto i partner dello stesso sono molti: Il Politecnico di Torino, l'Universtà di Bari, DS Graphic Engineering S.r.l, STRADE S.r.l, Linear System.
In particolare, il ruolo che assume il Politecnico di Torino è quello della Ricerca e Sviluppo della piattaforma in stretta collaborazione con l'azienda Linear System. Il team del Politecnico è composto da tre membri: Pietro Cilluffo, Giulia Meletti e Martino Massa.
Il processo di sviluppo si è attenuto il più possibile alle regole di riferimento della metodologia Agile, in modo da portare avanti il lavoro prettamente a distanza nella maniera più efficace possibile sia tra noi tesisti del Politecnico sia con l'azienda Linear System. Il lavoro del Politecnico previsto durante il periodo che concerne il lavoro di tesi riguarda due macro-periodi che possiamo suddividere in:

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

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

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

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

Oltre l'intrattenimento che è una componente fondamentale dell'evento, la digitalizzazione sta diventando una componente fondamentale non solo per rendere l'esperienza da parte dell'organizzatore e/o del partecipante unica ma anche grazie alla raccolta di dati, renderlo un'esperienza unica ed ad hoc per ogni singolo organizzatore/partecipante. Secondo alcuni studi riportati nell'analisi fatta nel documento del progetto DME:

> *"Solo per conferenze aziendali ed eventi per la promozione delle aziende, ogni anno vengono spesi a livello globale circa 512 Miliardi di dollari, mentre il valore complessivo di mercato di settore supera, annualmente, i 3.000 miliardi di dollari. Sempre nel settore degli eventi aziendali in Italia si è registrata una crescita, dopo la crisi del 2008, a partire dal 2014. Gli investimenti delle aziende in eventi sono cresciuti raggiungendo un volume di spesa complessivo pari a 819 milioni di euro nel 2016 e 852 milioni nel 2017. Le previsioni per i prossimi anni sono di crescita costante stimando che per il 2020 gli investimenti in eventi supereranno il miliardo*

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

Risulta evidente che il mercato in questo ambito è in 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

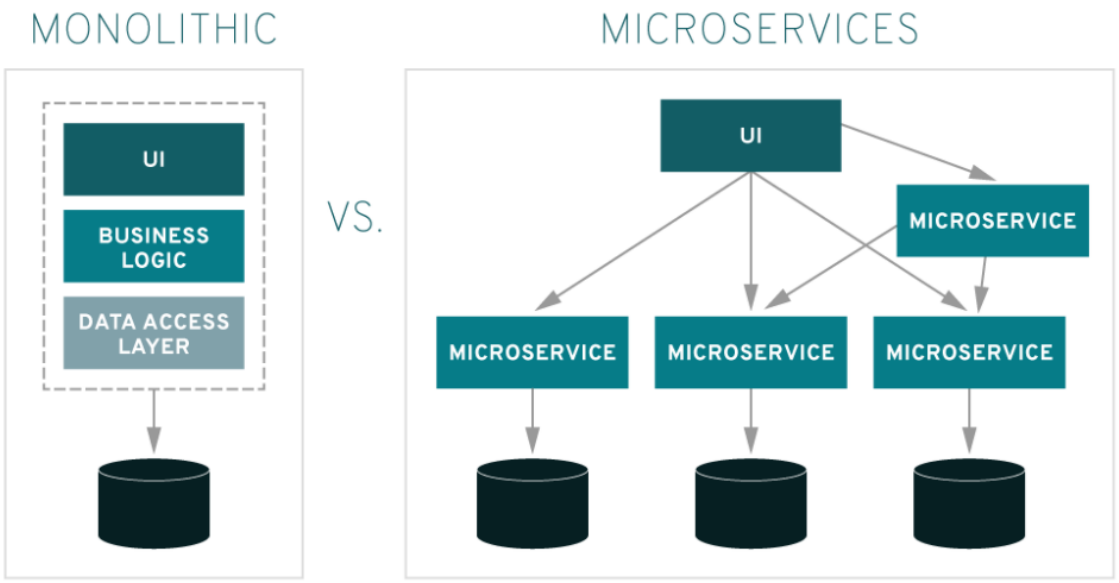


Figura 2.1: confronto tra schema monolitico e schema microservizi[1]

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.

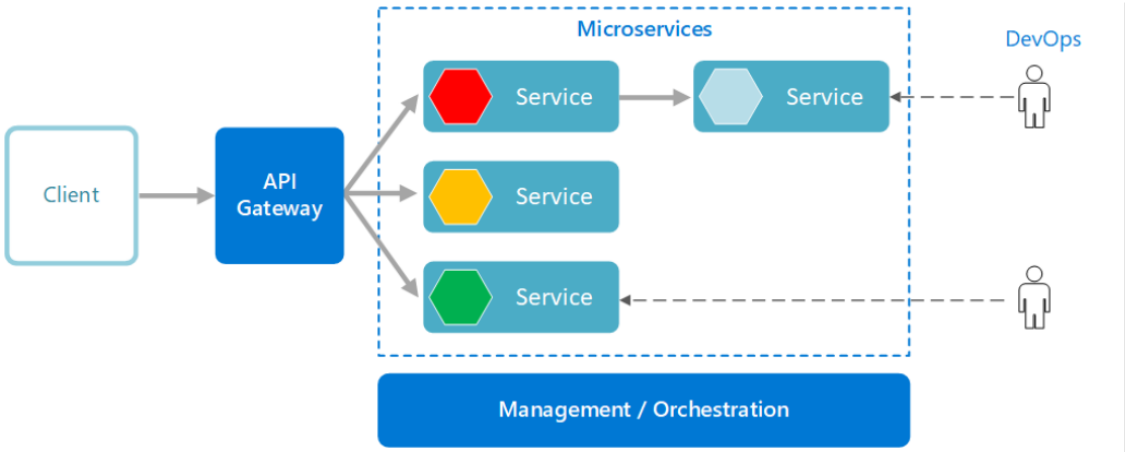


Figura 2.2: schema microservizio[2]

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:

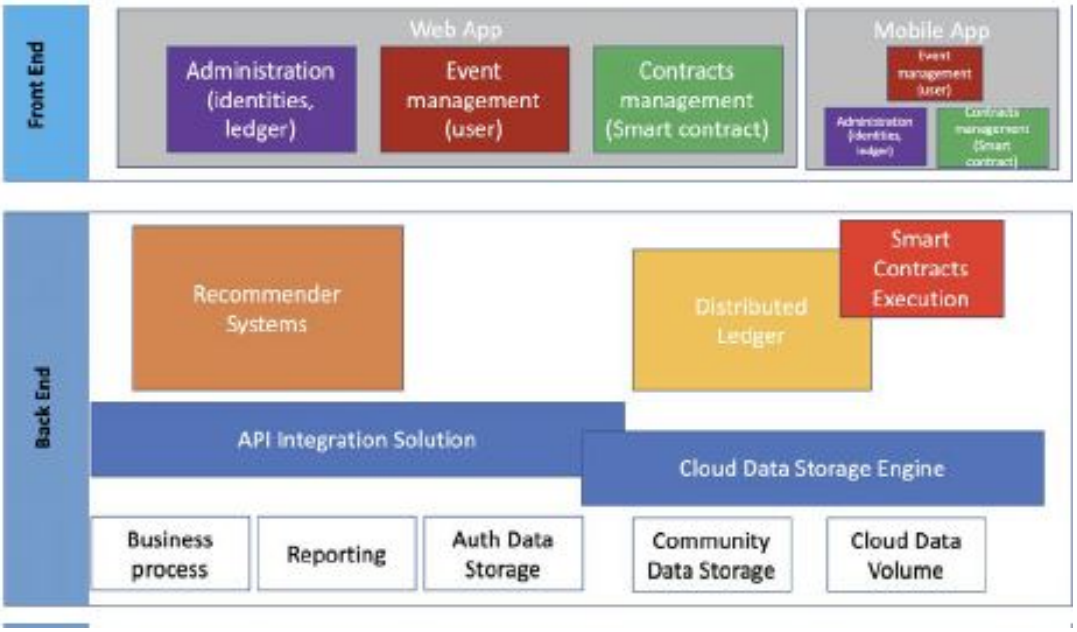


Figura 3.1: Schema a blocchi della piattaforma DME

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 per le varie tipologie di dispositivi: desktop, tablet, 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 Distributed Ledger per l'implementazione degli smart contract, esso è messo a disposizione mediante un servizio che espone delle API apposite, al fine di registrare le transazioni e i pagamenti tra fornitore e utente.

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:**

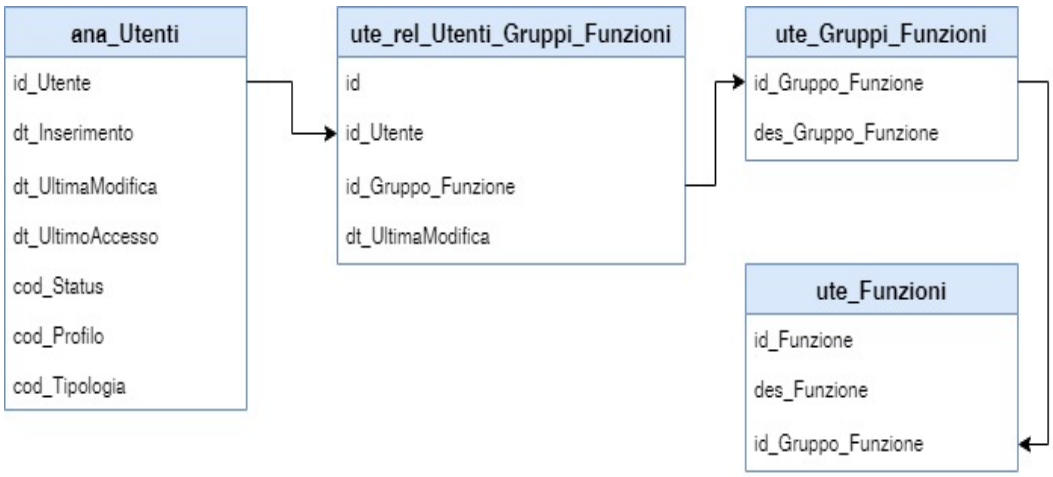


Figura 3.3: Microservizio Utente

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:**

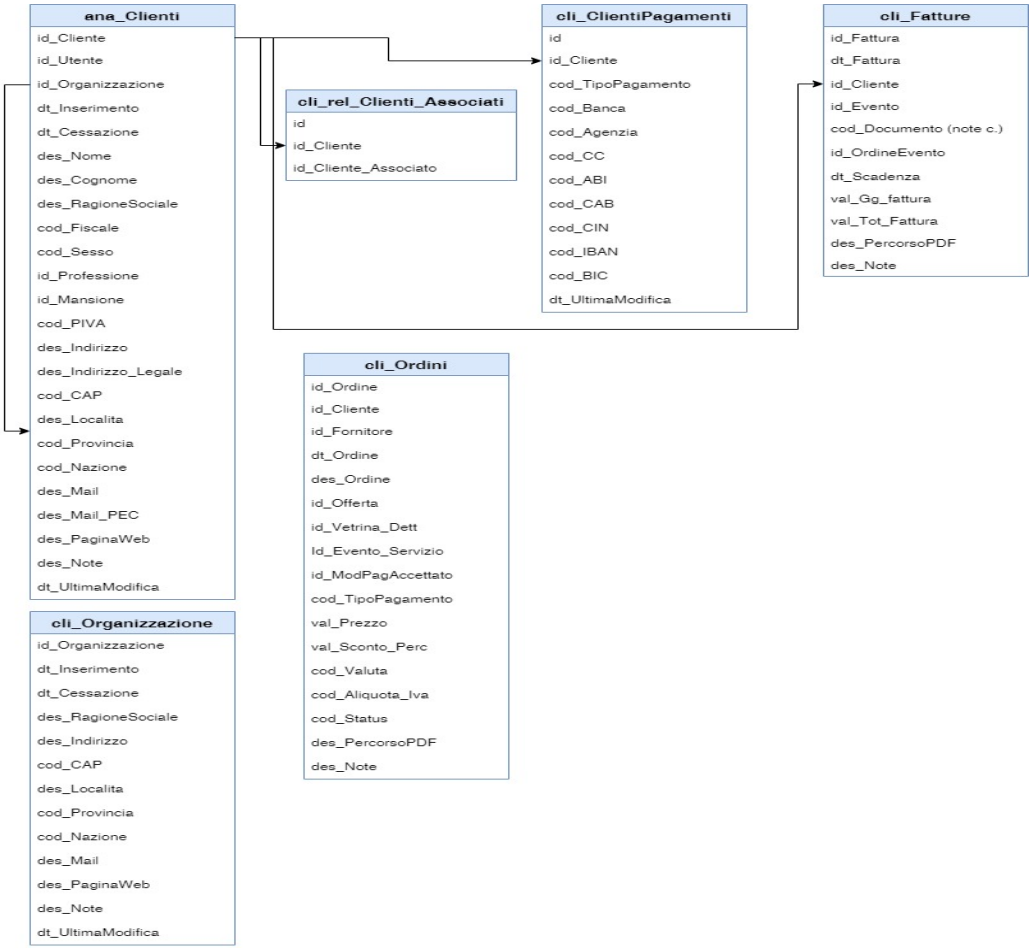


Figura 3.4: Microservizio Cliente

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:**

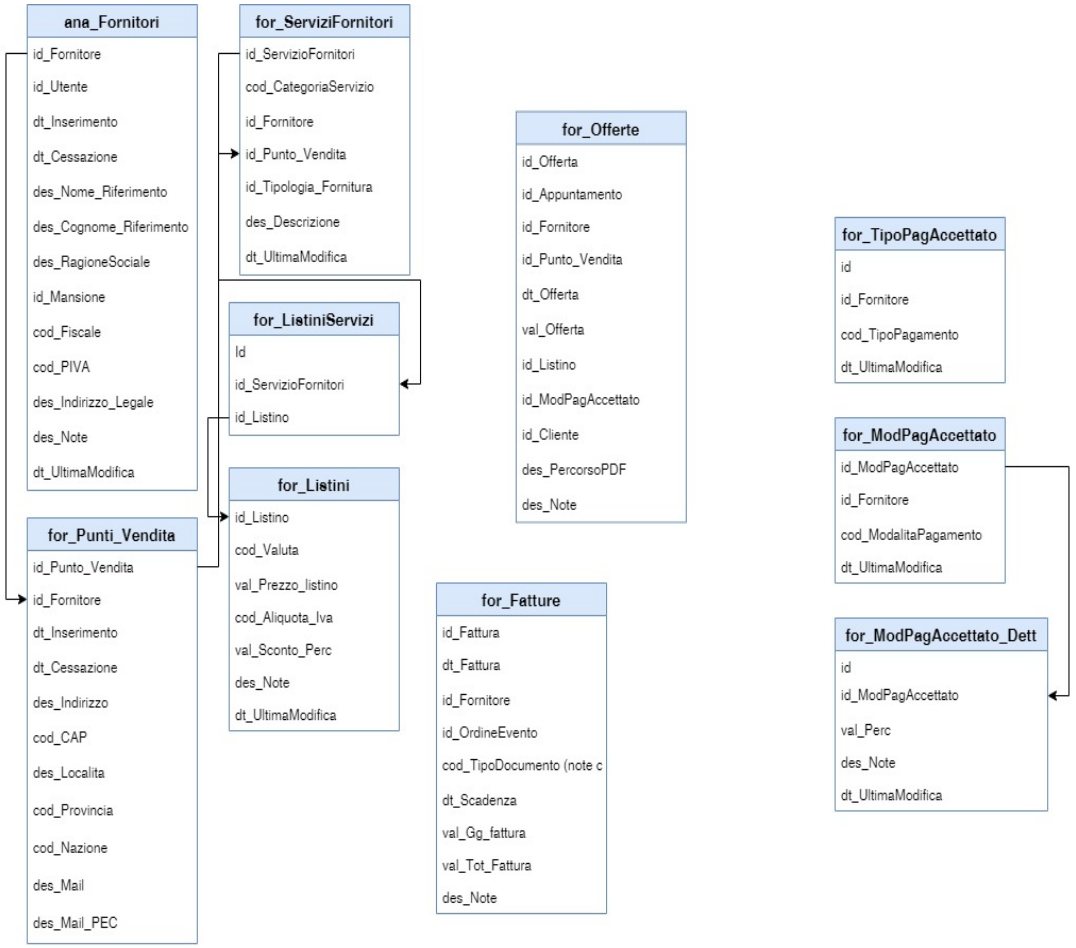
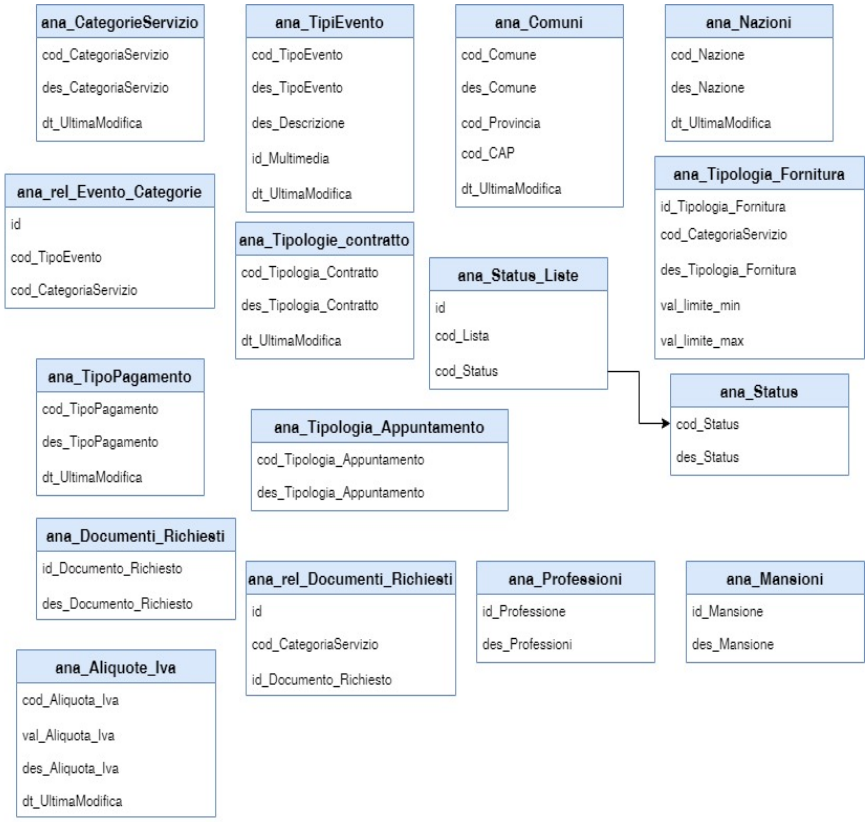


Figura 3.5: Microservizio Fornitore

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:**

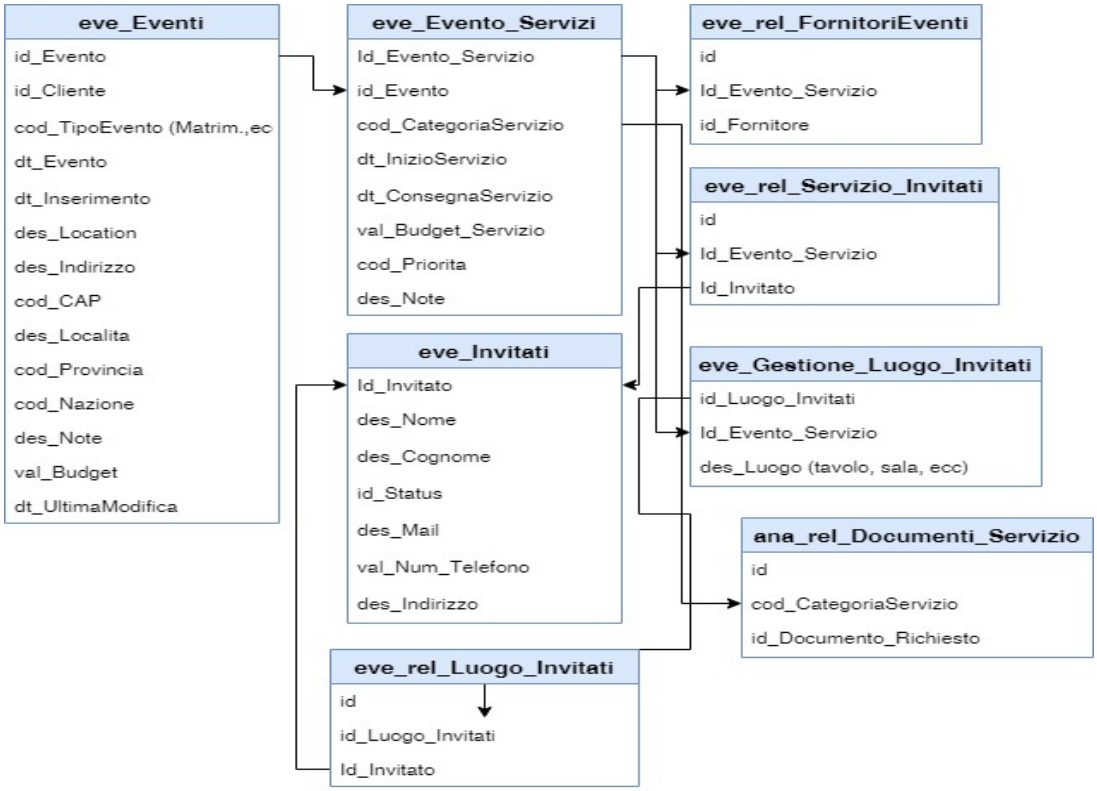


Figura 3.7: Microservizio Eventi

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:**

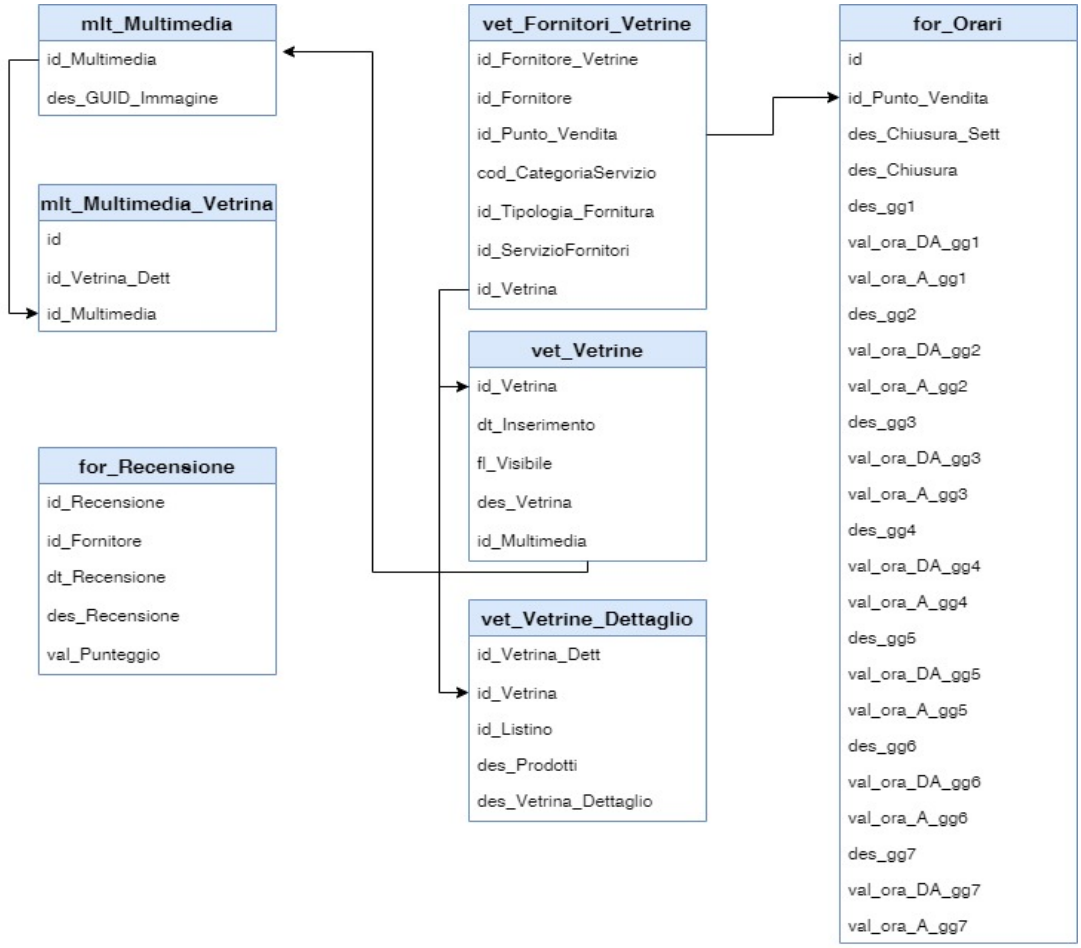


Figura 3.9: Microservizio Vetrine

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:**

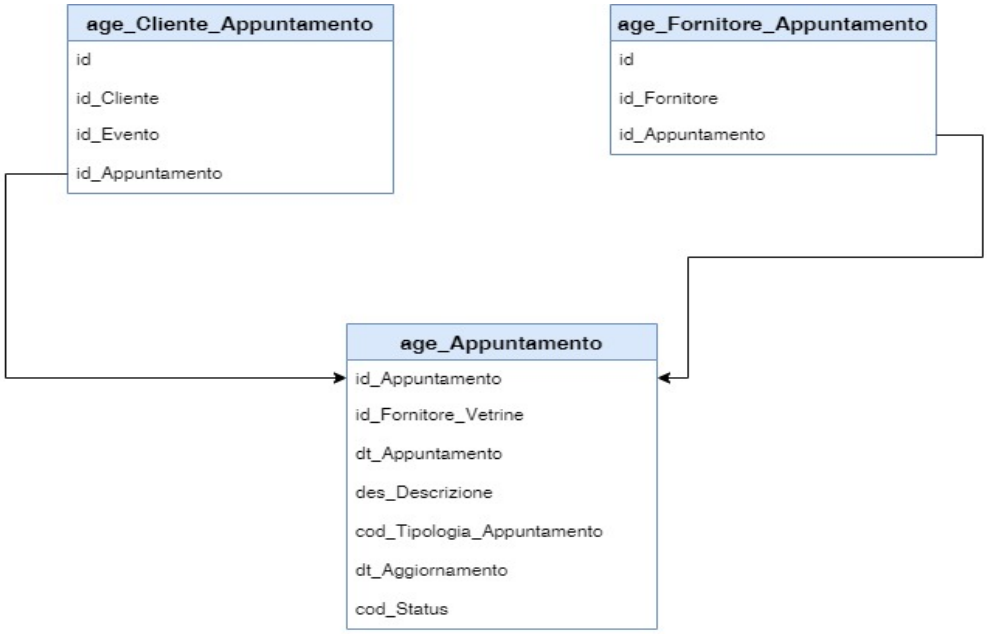


Figura 3.10: Microservizio Agenda

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.

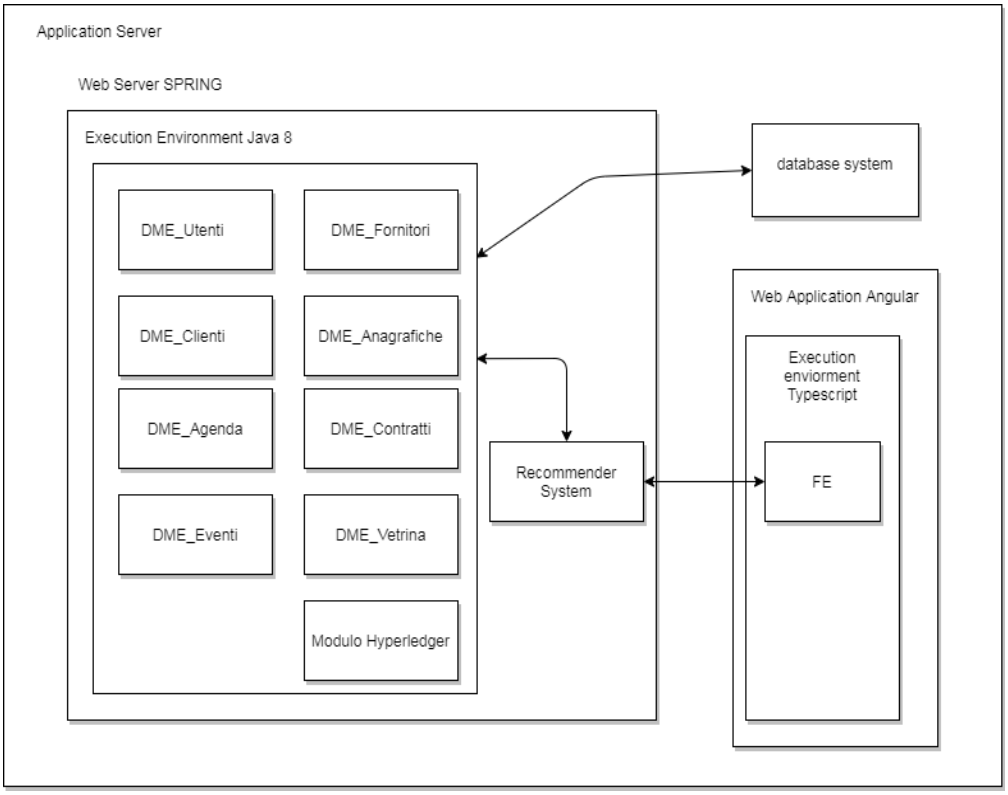


Figura 3.11: Deployment diagram

3.2 Layer Backend

Il core della web application nonchè 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.

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,l' aggiunta di un attore

individuato come amministratore che potrebbe interagire con il sistema effettuando 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 gli attori.
Nel nostro caso il context diagram di riferimento è sotto riportato:

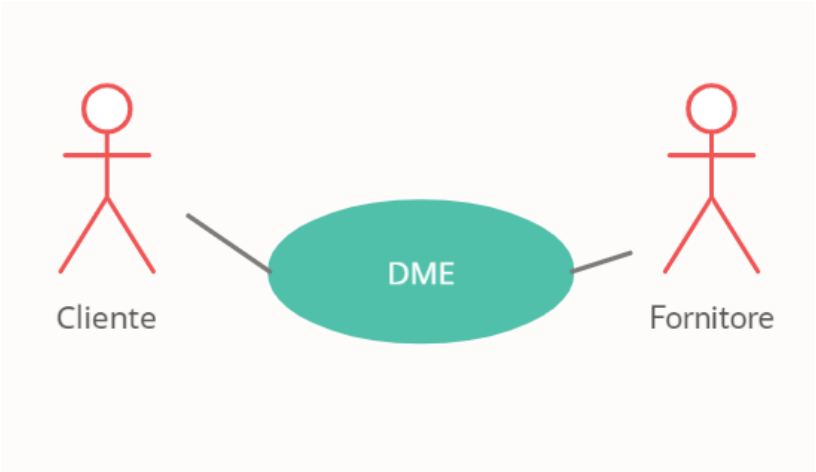


Figura 4.1: Context Diagram

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 categorie di evento, cercando ove possibile di renderlo il più idoneo possibile alla user experience.
Per fare ciò si è dunque deciso di prendere in considerazione un particolare caso d'uso, corrispondente ad un determinato tipo di evento, in modo da poterlo analizzare, studiare ed implementare con gli accorgimenti necessari per renderlo caratteristico. Nello specifico è stato preso 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 prototipo valido per molte altre tipologie di evento .
La piattaforma fornirà validi strumenti per l'organizzazione e permetterà la connessione tra i diversi attori della piattaforma, in maniera real time. Si potrà 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 per visualizzare 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. Il cliente potrà ricercare all'interno delle diverse categorie, inviare una richiesta, confrontare le diverse risposte e scegliere quello migliore. Tutta la trattativa cliente-fornitore potrà essere eseguita con il supporto della nostra piattaforma. Infine saranno presenti diversi tool di gestione: lista degli impegni (esempio: prenotazione location), budget (costo finale/ costo pagato) ed i fornitoriprenotati (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 avverranno nei mesi a seguire, visualizzerà il costo finale per l'organizzazione di tutto il matrimonio. A seguire decidono di iniziare il percorso di selezione e validazione dei diversi fornitori guidati dall'Agenda personalizzata del loro matrimonio. Tra le prime voci da scegliere c'è la chiesa, e con il portale Francesca riesce a gestire la condivisione dei diversi documenti, atti della cresima, nulla osta, tra le diverse parrocchie. Nei giorni successivi Francesca vuole iniziare a vedere le diverse location nelle quali organizzare il proprio ricevimento e così inizia a visualizzare la lista delle location su Roma, sul portale sono suddivise per categoria: Ville, Agriturismi, Hotel, Ristoranti, Castelli, Spiagge, etc.; selezionando arriva ad una decina di location che le piacciono e vede il dettaglio di ognuna: foto 3D di dettaglio, video, virtual tour, spazi e capienza, servizi offerti, possibilità di alloggio, posizionamento su mappa, eventuale esclusiva di alcuni fornitori, costo di affitto della struttura, n° massimo di invitati, ambienti, recensioni di altre coppie di sposi. Decide di inviare una richiesta di preventivo alle 10 location prescelte, tramite l'app. Ricevute le risposte dalle diverse location valuta la più indicata per il suo matrimonio e conferma la prenotazione per il 29 Maggio al Castello di Tor Crescenza. Il contratto viene archiviato nella sessione dedicata e l'acconto viene pagato sempre tramite app. Successivamente, sempre in base alla scaletta dell'agenda del matrimonio, precedentemente settata, passa alla prenotazione di: catering, allestimenti floreali, fotografo e video, bomboniere, partecipazioni, musica, noleggio auto, animazione adulti e bambini, noleggio arredi e tensostrutture, decorazioni varie, lista di nozze, wedding planner, torte nuziali, abito da sposa, abito da sposo, acconciatura, trucco, fedi. Per ognuno di questi fornitori ha la possibilità di vedere i diversi dettagli e contattare il fornitore per avere maggiori chiarimenti. Si susseguono mesi di ricerche, selezioni ed anche sopralluoghi e prove sul posto. Attraverso la piattaforma Francesca può sempre tenere sotto controllo attività fatte e da fare e aggiornare via via l'agenda con la Finita la prima fase di scelta e prenotazione dei fornitori, contemporaneamente Francesca ha bisogno di gestire gli invitati e quindi li aggiunge*

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

4.3.3 Use Case Diagram

Figura 4.2: Use case diagram

4.3.4 Use case

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

1. Creazione account:

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

Tabella 4.3: Use Case Creazione Account

2. Gestione creazione evento privato:

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

Tabella 4.4: Use Case Gestione creazione evento privato

3. Gestione attività stima costo:

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

Tabella 4.5: Use Case Gestione stima costo

4. Gestione Agenda:

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

Tabella 4.6: Use Case Gestione agenda

5. Gestione attività prenotazione chiesa:

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

Tabella 4.7: Use Case prenotazione chiesa

6. Gestione attività prenotazione ricevimento:

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

Tabella 4.8: Use case relativo alla prenotazione del ricevimento

7. Gestione contratti:

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

Tabella 4.9: Use case relativo alla gestione dei contratti

8. Gestione invitati:

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

Tabella 4.10: Use case relativo alla gestione degli invitati

4.4 Requisiti funzionali

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

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

Nella piattaforma DME i requisiti funzionali analizzati sono:

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 anche 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 analizzati sono:

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

Tecnologie utilizzate nel FrontEnd

In questo capitolo verranno descritti gli strumenti tecnici utilizzati per lo sviluppo del Front End. In particolare, il framework utilizzato per l'implementazione è Angular, i linguaggi utilizzati al suo interno sono Typescript, HTML e CSS. Inoltre, per migliorare l'efficienza dell' applicazione, è stata utilizzata la libreria RxJS, fortemente integrata in Angular. Infine per la comunicazione tra Front End e i moduli di Back End e Distributed Ledger è stato utilizzato lo stile architetturale REST.

5.1 Angular

Angular è un framework per lo sviluppo di applicazioni web single page. Le SPA (Single Page Application) sono applicazioni web o un sito web che può essere usato o consultato su una singola pagina web con l'obiettivo di fornire una esperienza utente più fluida e simile alle applicazioni desktop dei sistemi operativi tradizionali. In un'applicazione su single page tutto il codice necessario (HTML, JavaScript e CSS) è recuperato in un singolo caricamento della pagina. Le risorse necessarie sono caricate dinamicamente e aggiunte alla pagina quando necessario, di solito in risposta ad azioni dell'utente. La pagina non si ricarica in nessun punto del processo, né lascia il controllo in un'altra pagina.

I casi in cui realizzare una Single Page Application come soluzione ideale sono spesso marcati da caratteristiche ricorrenti:

- qualora sia necessaria un'applicazione dinamica, sempre disponibile in real-time
- in tutti i casi in cui l'interazione da parte dell'utente e la User Experience sia al centro del progetto

- qualora vi siano diversi stati condivisi tra più schermate

L'intenzione originaria di Angular era quella di creare uno strumento semplice e rapido per lo sviluppo di applicazioni capaci di girare su qualunque piattaforma e facilmente fruibili da desktop, tablet o smartphone. La combinazione di Angular e Bootstrap consente oggi di creare applicazioni perfettamente responsive e capaci di ottimizzare il proprio layout in funzione del dispositivo utilizzato. Con Angular le applicazioni vengono eseguite interamente dal browser e questo implica innanzitutto un risparmio di tempo netto ogni qualvolta si verifica un'azione da parte di un utente. L'architettura Angular consente in questo modo di creare applicazioni complesse e sofisticate, ma al contempo leggere, veloci e facilmente fruibili.

In sintesi, le caratteristiche principali del framework sono:

- *Estende l'HTML*: con Angular si può utilizzare HTML come XML, offrendo infinite possibilità per tag e attributi attraverso la creazione di custom-tag che combinano presentazione e logica.
- *Data Binding and Dependency Injection*: le variabili possono essere sincronizzate tra view e model, inoltre le dipendenze sono automaticamente iniettate quando necessario.
- *Rest*: è possibile comunicare facilmente con il server e ottenere i dati necessari per interagire con le tue pagine web.
- *Cross Platform*: con Angular è possibile creare applicazioni e riutilizzare il codice e le abilità per creare app per qualsiasi target di distribuzione (per web, web mobile, mobile nativo e desktop nativo).

Angular segue il pattern MV * che è un ibrido dei pattern MVC e MVVM. In precedenza abbiamo esaminato il modello MVC. Ad alto livello l'architettura di entrambi i modelli è relativamente simile, come mostrato nel diagramma che segue:

Figura 1.9: Architettura MV *

Il nuovo concetto qui è il ViewModel, che rappresenta il codice che collega la View al Model o Servizio. In Angular, questo collegamento è noto come binding. Mentre i framework MVC come Backbone o React devono chiamare un metodo di rendering per elaborare i loro model HTML, in Angular questo processo è semplice e trasparente per lo sviluppatore. Il binding è ciò che differenzia un'applicazione MVC da un'applicazione MVVM. I vantaggi dell'utilizzo di framework di questo tipo sono enormi:

- **Separation Of Concerns:** possiamo costruire applicazioni frontend con molto meno codice rispetto alle precedenti tecnologie di generazione. Questo

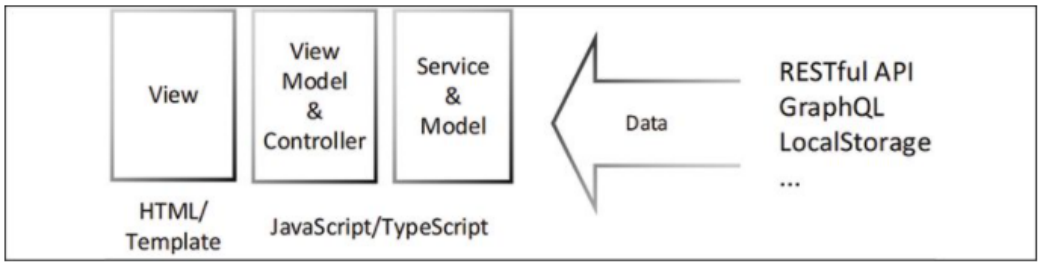


Figura 5.1

perché non dobbiamo scrivere un codice che sincronizza manualmente il modello e la vista in quanto quel codice è generato automaticamente per noi. Il codice che dobbiamo scrivere sarà molto più leggibile e mantenibile, a causa della netta separazione delle preoccupazioni tra Model e View.

- **Modello trasparente per visualizzare la sincronizzazione:** la sincronizzazione tra Model e View non è solo trasparente ma è ottimizzata in un modo che non è possibile ottenere in pratica manualmente.

- **Prestazioni dell'interfaccia utente:** la vista viene modificata generando direttamente alberi di oggetti DOM in un modo compatibile con crossbrowser, invece di generare HTML e poi passandolo al browser per l'analisi: questo ignora efficacemente il file fase di analisi del tutto. L'HTML deve ancora essere analizzato, ma viene eseguito solo una volta per modello da Angular e dal browser ogni volta. L'analisi viene eseguita in tempo di avvio dell'applicazione (Just In Time o JIT) o in anticipo (AOT) quando si crea l'applicazione. E questo è uno dei motivi principali per cui vale la pena avere più ambiente di sviluppo avanzato come la CLI, perché con esso possiamo avere tutte queste caratteristiche e vantaggi che lavorano in modo trasparente fuori dagli schemi. Inoltre, la generazione e la modifica della vista vengono eseguite in un file ottimizzato, il che significa che solo le parti dell'albero DOM che hanno i dati modificati saranno interessati, mentre il resto della pagina rimane il file stesso.

5.1.1 Confronto con altri framework e librerie

Oggi praticamente qualsiasi framework realizza e mette a disposizione degli utenti i vantaggi sopra descritti. E' invece vero che Angular in particolare offre una qualità avanzata in termini di mantenibilità, flessibilità e strutturazione avanzate delle applicazioni. Essendo Angular un vero e proprio framework e non una semplice

libreria (come React, ad esempio), punta a dare moltissimi strumenti e pattern sia semplici che avanzati per costruire applicazioni di ogni tipo, mantenendo sempre, se usato con conoscenza, un codice pulito, ordinato e mantenibile. Grazie agli strumenti di contorno e al forte supporto della comunità si riesce a integrare Angular in qualsiasi contesto. Oltre al classico contesto WEB, Angular si dimostra oltremodo efficace sia come applicazione mobile (grazie a wrapper come Ionic e simili), sia in app desktop (grazie a Electron ecc). Si avvantaggia poi, e non poco, di moltissime librerie per integrarlo con ulteriori servizi con estrema facilità (Google, AWS, strumenti per l'offline ecc...).

5.1.2 Architettura Angular

L'architettura di un'applicazione Angular si basa su alcuni concetti fondamentali. Gli elementi costitutivi di base del framework Angular sono componenti Angular organizzati in NgModules. NgModules raccoglie il codice correlato in set funzionali. Un'app Angular è definita da un insieme di NgModules, in particolare ha sempre almeno un modulo radice che consente il bootstrap e diversi moduli di funzionali. I componenti definiscono le viste, che sono insiemi di elementi dello schermo che Angular può mostrare e modificare in base alla logica e ai dati del programma. A loro volta i componenti utilizzano servizi che forniscono funzionalità specifiche non direttamente correlate alle visualizzazioni. Moduli, componenti e servizi sono classi che utilizzano decoratori: essi hanno la funzione di contrassegnare il loro tipo e forniscono metadati. I metadati associati a una classe component definiscono il modello della vista. Un modello combina HTML ordinario con direttive Angular e binding markup che consentono ad Angular di modificare l'HTML prima di renderlo per la visualizzazione. Mentre i metadati per una classe di servizio forniscono le informazioni necessarie ad Angular per renderlo disponibile ai componenti tramite DI (dependency injection).

Un'applicazione Angular in genere definisce molte visualizzazioni, disposte gerarchicamente; per navigare tra le viste si può utilizzare il servizio Router. Il diagramma seguente mostra come questi componenti di base sono correlati.

In particolare, l'insieme di un component e di un model definisce una vista Angular. Un decoratore su una classe component aggiunge i metadati, incluso un puntatore al template associato. Direttive e binding markup nel template di un component modificano la view in base alla logica e ai dati del programma. Infine, il dependency injector fornisce servizi a un component, come ad esempio il servizio router che consente di definire la navigazione tra le varie schermate. [3]

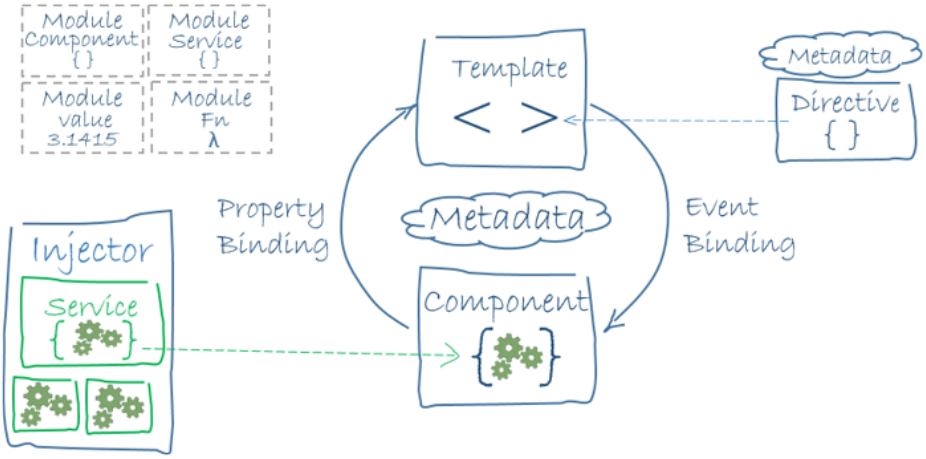


Figura 5.2: Relazione tra elementi di un' applicazione Angular [3]

5.1.3 Typescript

TypeScript è il linguaggio principale per lo sviluppo di applicazioni Angular. Questo linguaggio è un'estensione di JavaScript, per essere precisi un superset di EcmaScript 5, che affronta le carenze di Javascript nello sviluppo e la manutenzione di applicazioni di grandi dimensioni.

TypeScript arricchisce JavaScript con un sistema di moduli, classi, interfacce e un sistema di tipi statici.
Il compilatore TypeScript controlla i programmi TypeScript ed emette JavaScript in modo che i programmi possano essere eseguiti immediatamente in una vasta gamma di ambienti di esecuzione. Il sistema di tipi TypeScript comprende una serie di concetti e costrutti avanzati: l'equivalenza del tipo strutturale (piuttosto che l'equivalenza del tipo per nome), i tipi per la programmazione basata sugli oggetti, la digitazione graduale, la sottotipizzazione dei tipi ricorsivi e il tipo operatori. Tutte queste caratteristiche dovrebbero contribuire a un armoniosa esperienza di programmazione.

Quando JavaScript è stato inventato ci si aspettava che venisse utilizzato per brevi frammenti di codice incorporati in una pagina Web. Nel corso del tempo, tuttavia, JS è diventato sempre più popolare e gli sviluppatori Web hanno iniziato a utilizzarlo per creare esperienze interattive. Sui siti Web moderni, infatti, il browser esegue spesso applicazioni che si estendono su centinaia di migliaia di righe di codice.

Caratteristiche Typescript:

- **Controllo statico del tipo:** TypeScript verifica la presenza di errori in un programma prima dell'esecuzione e lo fa in base al tipo di valori,
- **Tipi:** TypeScript è un superset tipizzato, il che significa che aggiunge regole su come possono essere utilizzati diversi tipi di valori.
- **Comportamento in runtime:** in linea di principio, TypeScript non modifica mai il comportamento di runtime del codice JavaScript. Mantenere lo stesso comportamento di runtime di JavaScript è una premessa fondamentale di TypeScript perché significa che si può passare facilmente tra i due linguaggi senza preoccuparsi di sottili differenze che potrebbero far smettere di funzionare il programma.
- **Tipi cancellati:** in parole povere, una volta che il compilatore di TypeScript ha finito di controllare il codice, cancella i tipi per produrre il codice "compilato" risultante.
- **TypeScript non fornisce alcuna libreria di runtime aggiuntiva:** i programmi useranno la stessa libreria standard (o librerie esterne) dei programmi JavaScript.

[4][5]

5.1.4 Reactive Development Paradigm

Nella documentazione ufficiale di RxJS è possibile trovare la dedinizione di questo strumento:

> *RxJS is a library for composing asynchronous and event-based programs by using observable sequences. It combines the Observer pattern with the Iterator pattern and functional programming to fill the need for an ideal way of managing sequences of events.*[6]

Tradotto in italiano:

> *RxJS è una libreria per la composizione di programmi asincroni e basati su eventi utilizzando sequenze osservabili. Combina il pattern Observer con il pattern Iterator e la programmazione funzionale per soddisfare la necessità di un modo ideale di gestire sequenze di eventi.*

RxJS è una libreria per gestire eventi asincroni tramite l'utilizzo del concetto di Observable e del paradigma funzionale (reattivo). Quando parliamo di eventi asincroni non si fa solo riferimento a chiamate HTTP ma anche a timer o eventi generati dal DOM tramite iterazioni utente.

I costrutti su cui si basa RxJS sono i seguenti:

- **Observable:** una sorta di stream di informazioni che contiene dati sincroni e asincroni. Si può sottoscrivere questo stream (tramite il metodo subscribe) e ricevere passivamente i dati non appena sono emessi.

- **Observer:** un Observer può essere definito come un "consumer" che riceve valori emessi da un Observable. Gli Observer sono semplici oggetti con 3 callbacks: next, error, complete che ricevono notifiche dall'Observable.

- **Subscription:** una Subscription è un riferimento all'esecuzione di un Observable ed è utilizzato, principalmente, per terminare la sua esecuzione tramite il metodo unsubscribe. Ogni observer ha un'esecuzione indipendente dall'Observable rispetto alle altre e questa modalità di funzionamento viene definita "unicast", ovvero due sottoscrizioni differenti allo stesso Observable sono indipendenti l'una dall'altra e non interferiranno tra di loro.

- **Operators:** funzioni "pure" che ricevono un Observable come input e ne restituiscono uno come output. Queste funzioni, chiamate operatori, sono utili per manipolare i valori forniti dagli Observable o, ad esempio, per "trasformare" un Observable da un tipo ad un altro e possono essere facilmente concatenati tramite un approccio dichiarativo.

- **Subject:** sono utilizzati per il "multicasting", dunque offrono la possibilità di condividere l'esecuzione dell'observer e gestire sottoscrizioni multiple allo stesso Observable.

L'utilizzo di RxJS è complesso ma se usato con coscienza porta a notevoli vantaggi:

- L'utilizzo di un unico approccio per gestire dati sincroni e asincroni rappresenta un gran vantaggio.

- Il paradigma funzionale riduce la possibilità di errori nel codice e, in connubio con RxJS, rende il codice molto più espressivo e conciso.

- RxJS è una soluzione adottata in diversi contesti, sia nel mondo front-end che back-end. Viene infatti utilizzato anche in Node, Java, React (Redux Observable) e in moltissimi altri contesti.

- Il framework Angular è totalmente basato sul concetto di Observable: HttpClient, gli @Output EventEmitter, le guardie e gli eventi del router, i Reactive Forms (e potrei continuare) hanno alla base questo paradigma.

[7]

5.2 REST

REST(Representational State Transfer) è uno stile architetturale per sistemi distribuiti.

REST ha risposto alla necessità della Internet Engineering Task Force (IETF) di un modello di come il Web dovrebbe funzionare. È un modello idealizzato delle interazioni all'interno di un'applicazione Web complessiva. [8]

Roy T. Fielding definisce REST in «La separazione REST client-server degli interessi semplifica l'implementazione del componente, riduce la complessità della semantica del connettore, migliora l'efficacia dell'ottimizzazione delle prestazioni ed aumenta la scalabilità di componenti server puri. I vincoli di sistema a strati permettono di introdurre intermediari — proxy, gateway e firewall — in vari punti della comunicazione senza cambiare le interfacce tra i componenti, consentendo loro di assistere nella traduzione della comunicazione o migliorare le prestazioni tramite cache condivisa di larga scala. REST consente l'elaborazione intermedia vincolando i messaggi ad essere auto-descrittivi: l'interazione è priva di stato tra le richieste, i metodi di base ed i tipi di media sono utilizzati per indicare la semantica e scambiare informazioni, le risposte indicano esplicitamente la possibilità di memorizzare nella cache.» [9]

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. 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. [10]

Capitolo 6

Progettazione Front End

6.1 Web Application

Il Front End per il progetto Digital Management Events sarà costistuito da una Web Application. Un'applicazione Web è un'applicazione fruibile attraverso internet da un browser Web. In appena un decennio, il Web si è evoluto dall'essere un repository di pagine utilizzate principalmente per l'accesso a informazioni statiche, per lo più scientifiche, a una potente piattaforma per lo sviluppo e la distribuzione di applicazioni. Nuove tecnologie Web, linguaggi e metodologie consentono di creare applicazioni dinamiche che rappresentano un nuovo modello di cooperazione e collaborazione tra un gran numero di utenti. I futuri sviluppi nelle applicazioni Web saranno guidati dai progressi nella tecnologia dei browser, nell'infrastruttura Web Internet, negli standard dei protocolli, nei metodi di ingegneria del software e nelle tendenze delle applicazioni. [11]

Attualmente sono disponibili due approcci generali per la creazione di applicazioni Web: applicazioni Web tradizionali che eseguono la maggior parte della logica dell'applicazione nel server e applicazioni a singola pagina (Spa) che eseguono la maggior parte della logica dell'interfaccia utente in un Web browser, comunicando con il server Web principalmente usando le API Web.
Le **app Web tradizionali** sono la migliore soluzione quando si hanno le seguenti necessità:

- **L'applicazione ha requisiti lato client semplici o di sola lettura:** Molte applicazioni Web vengono usate principalmente in modalità di sola lettura dalla maggior parte degli utenti. Le applicazioni di sola lettura (o di sola lettura) tendono a essere molto più semplici rispetto a quelle che gestiscono e manipolano una grande quantità di stato.
- **L'applicazione deve funzionare in browser senza supporto per JavaScript:** Le applicazioni Web che devono funzionare in browser con supporto

limitato o senza supporto per JavaScript devono essere scritte con i flussi di lavoro tradizionali per le app Web (o almeno devono essere in grado di attivare il comportamento corrispondente se necessario). Per il funzionamento delle applicazioni a pagina singola, JavaScript deve essere disponibile sul lato client. In caso contrario, le applicazioni a pagina singola non rappresentano la scelta ottimale.

Mentre, le **applicazioni a pagina singola** sono la migliore soluzione quando si hanno le seguenti necessità:

- **L'applicazione deve esporre un'interfaccia utente avanzata con numerose funzionalità:** Le applicazioni a pagina singola supportano funzionalità lato client complete, che non richiedono di ricaricare la pagina quando gli utenti eseguono operazioni o passano da un'area all'altra dell'app. Le applicazioni a pagina singola vengono caricate rapidamente recuperando i dati in background e le singole azioni dell'utente risultano più rapide, perché pagine vengono ricaricate completamente solo in casi sporadici. Le applicazioni a pagina singola supportano gli aggiornamenti incrementali, salvando documenti o moduli completati solo parzialmente senza richiedere all'utente di fare clic su un pulsante per inoltrare un modulo. Le applicazioni a pagina singola supportano comportamenti lato client complessi.

[12] La scelta progettuale per Digital Management Events è stata l'implementazione di una Single Page Application con l'utilizzo del framework Angular. Le SPA sono la miglior alternativa per lo sviluppo delle complesse funzionalità richieste dai requisiti e per la gestione degli stati. Con questo approccio è possibile una migliore User Experience in quanto i dati vengono recuperati dal server di backend in background.

6.1.1 Creazione dell'interfaccia utente composita basata su microservizi

L'architettura di microservizi spesso inizia con la gestione dei dati e della logica sul lato server, ma in molti casi l'interfaccia utente viene ancora gestita come Monolith. Un approccio più avanzato, denominato micro-front-end, consiste nel progettare l'interfaccia utente dell'applicazione in base ai microservizi. Ciò significa avere un'interfaccia utente composita generata dai microservizi, invece di avere i microservizi sul server e solo un'app client monolitica che utilizza i microservizi. Con questo approccio, i microservizi creati possono essere completati con la rappresentazione sia logica che visiva. La figura 6.1 illustra l'approccio più semplice in cui i microservizi vengono utilizzati da un'applicazione client monolitica. La figura è una rappresentazione semplificata che evidenzia la presenza di una sola interfaccia

utente client (monolitica) che utilizza i microservizi, concentrati solo sulla logica e sui dati e non sulla forma dell'interfaccia utente (HTML e JavaScript).

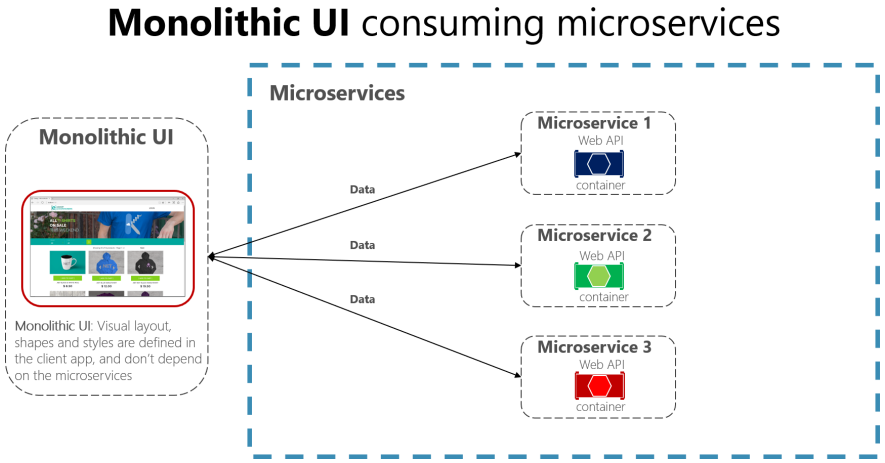


Figura 6.1: Applicazione dell'interfaccia utente monolitica che utilizza i microservizi back-end

Un'interfaccia utente composita viene invece generata con precisione ed è composta dai microservizi stessi. Alcuni dei microservizi gestiscono la forma visiva di aree specifiche dell'interfaccia utente. La differenza principale è che sono presenti componenti dell'interfaccia utente client (ad esempio, le classi TypeScript) basati su modelli e che l'elemento dell'interfaccia utente di data shaping ViewModel di tali modelli deriva da ogni microservizio. Durante l'avvio dell'applicazione client, ognuno dei componenti dell'interfaccia utente client (ad esempio, le classi TypeScript) si registra con un microservizio dell'infrastruttura in grado di fornire il ViewModel per un determinato scenario. Se la forma del microservizio cambia, anche l'interfaccia utente cambia. Questo approccio è semplificato perché potrebbero essere presenti altri microservizi che aggregano parti granulari basate su tecniche diverse.
Ogni microservizio di composizione dell'interfaccia utente dovrebbe essere simile a un gateway API di piccole dimensioni. Ma in questo caso ognuno è responsabile di una piccola area dell'interfaccia utente. Un approccio basato sull'interfaccia utente composita gestito dai microservizi può essere più o meno complesso a seconda delle tecnologie usate per l'interfaccia utente. [13]

Nel progetto DME è stato seguito, quando le informazioni dei diversi microservizi non necessitavano di essere intrecciate, un approccio micro-front-end in cui i singoli componenti dell'applicazione comunicano con i microservizi necessari alla visualizzazione dei dati.

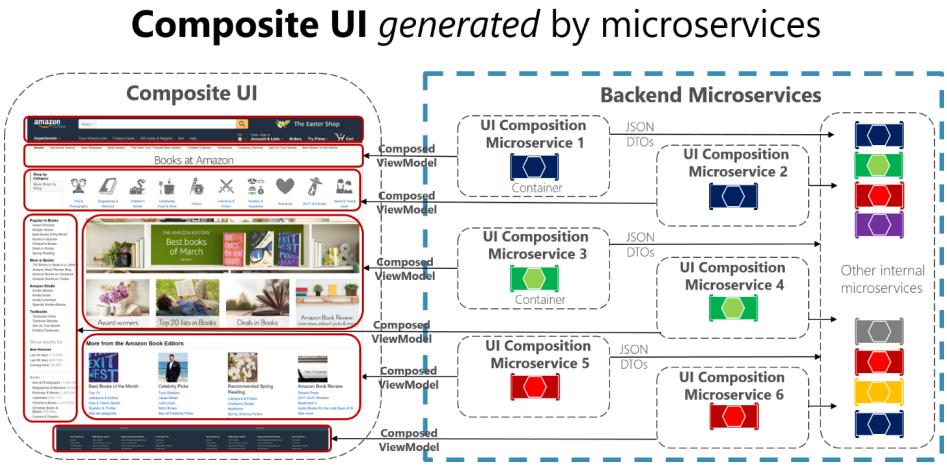


Figura 6.2: Esempio di applicazione con interfaccia utente composita modellata da microservizi back-end

6.2 Responsive Design

Il responsive design è una tendenza importante nello sviluppo web per soddisfare la diversità dei dispositivi utilizzati per la navigazione web. Recenti studi industriali dimostrano che la reattività delle applicazioni web influisce direttamente sulla soddisfazione del cliente e, di conseguenza, su metriche business-critical come entrate e tassi di conversione. Responsive web design significa costruire il layout dell'interfaccia web su griglie fluide in grado di adattarsi dinamicamente a diversi dispositivi di visualizzazione. A livello tecnico, ciò si ottiene utilizzando unità relative (percentuali o em) piuttosto che unità assolute (pixel o punti) per il dimensionamento degli elementi della pagina nonché le mediaquery CSS3 per applicare regole CSS diverse in base alla posizione e al floating degli elementi a seconda delle dimensioni della finestra del browser. [14]

Per soddisfare il requisito di adattabilità dei contenuti, in base al dispositivo, per il progetto DME sono stati utilizzati 3 diversi elementi

- CSS3 media queries
- Bootstrap
- Alcuni componenti Angular Material

Capitolo 7

Implementazione Front End

7.1 Architettura

In questa sezione sarà analizzata l'architettura front end della piattaforma DME e la navigazione. Le numerose funzionalità di ricerca e gestione implementate nell'applicazione web impongono che essa sia articolata in numerose schermate in cui ognuna di esse ha il compito di soddisfare una o più funzionalità. L'applicazione prevede l'esistenza di due stackeholder: Cliente e Fornitori. Per ognuno sono state studiate e sviluppate funzionalità diverse.

Le funzionalità a disposizione del Cliente sono:

- Registrazione
- Login
- consultazione e modifica profilo
- Creazione di un evento
- modifica evento
- definizione budget
- ricerca fornitori eventualmente filtrata per categoria
- visione vetrina fornitore selezionato, appuntamento e ordine
- consultazione agenda appuntamenti
- consultazione ordini
- definizione lista invitati

- definizione tavoli
- visione e gestione contratti

Le funzionalità previste per il fornitore invece sono:

- Registrazione
- Login
- consultazione e modifica profilo
- Creazione di un servizio
- modifica servizio e relativa vetrina
- gestione e consultazione appuntamenti
- gestione e consultazione ordini
- gestione e consultazione contratti

Nella figura sottostante è riassunta la relazione tra le diverse schermate/funzionalità in una struttura ad albero in modo da poterne analizzare la gerarchia.

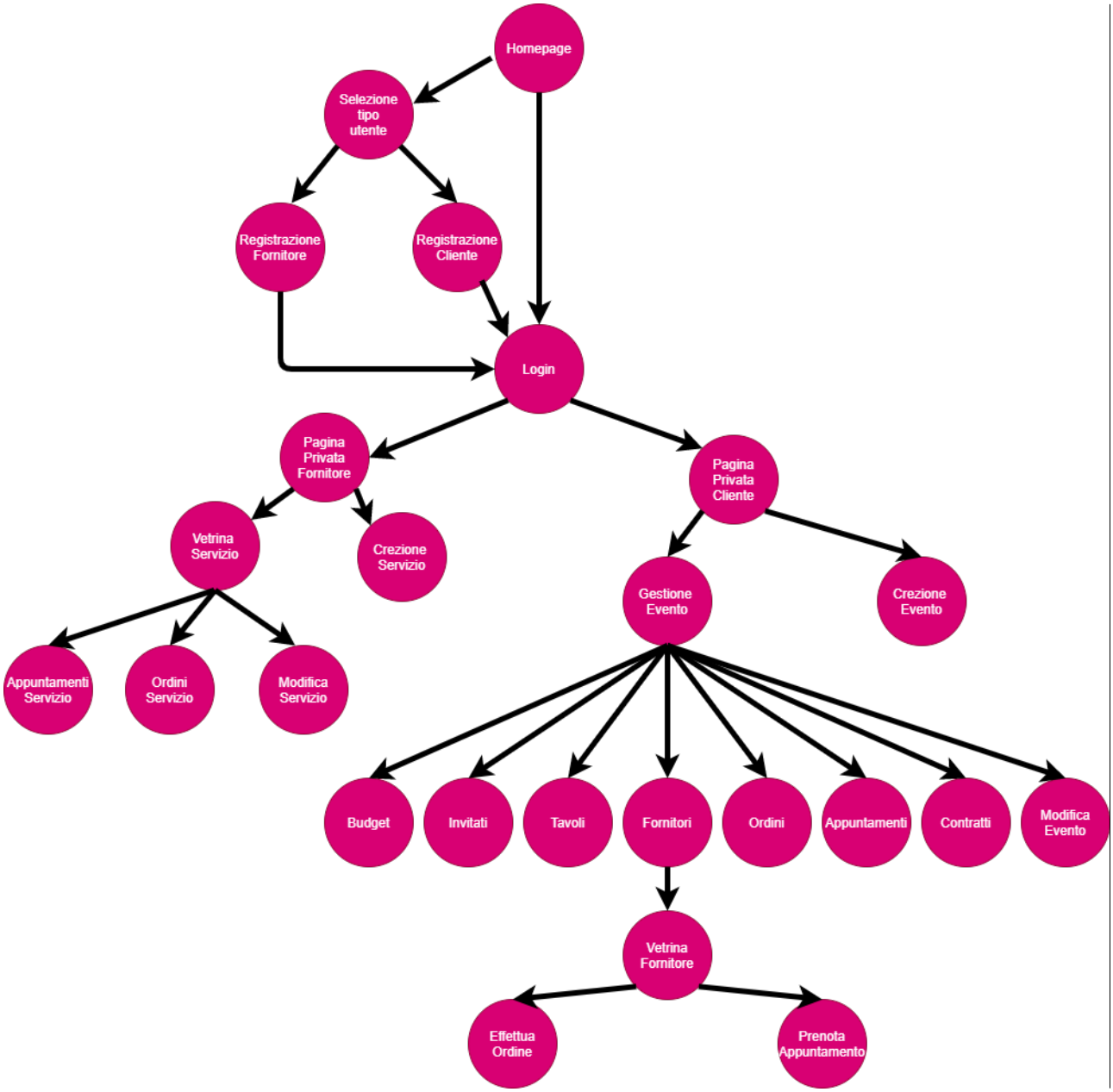


Figura 7.1: Architettura view

7.2 Navigazione

7.2.1 Navigazione generale

Homepage

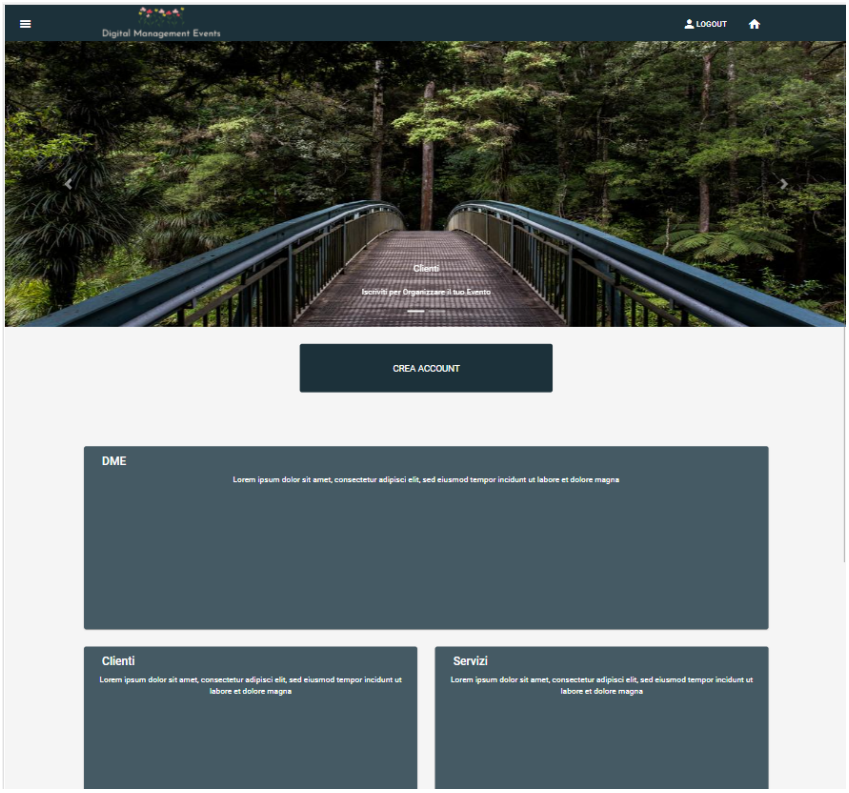


Figura 7.2: Homepage

L'homepage è la radice dell'applicazione. Da questa schermata è possibile proseguire con la registrazione mediante il tasto "Crea Account" oppure accedere alla pagina di login con l'apposito bottone presente nella navigation bar. La dashboard, situata nella parte inferiore della pagina, permetterà ai gestori di Digital Management Events di fornire alcune informazioni utili sulla piattaforma e catturare l'utente con una presentazione efficace.

Selezione tipologia utente

Una volta raggiunta questa schermata, mediante il tasto "Crea Account", presente sia nella homepage che nella pagina di log, l'utente dovrà scegliere se registrarsi con un profilo da cliente oppure da fornitore.

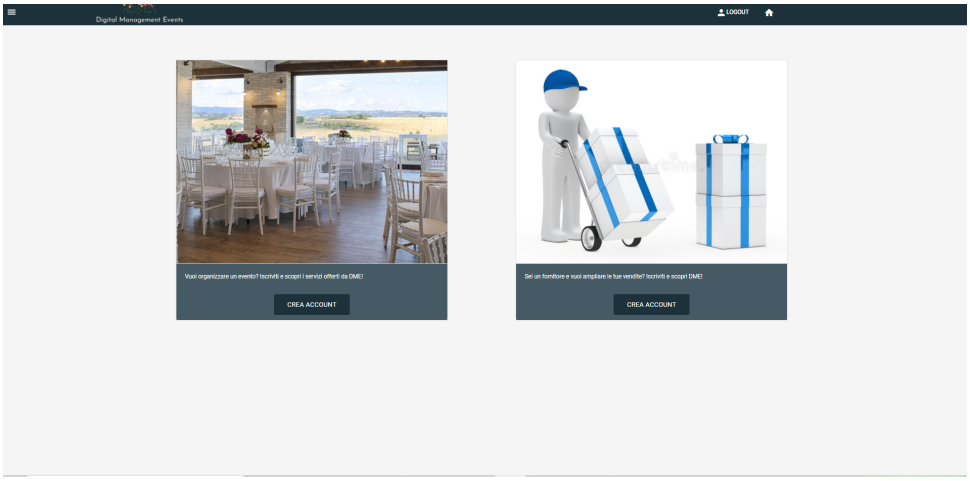


Figura 7.3: Selezione tipo utente

Registrazione cliente

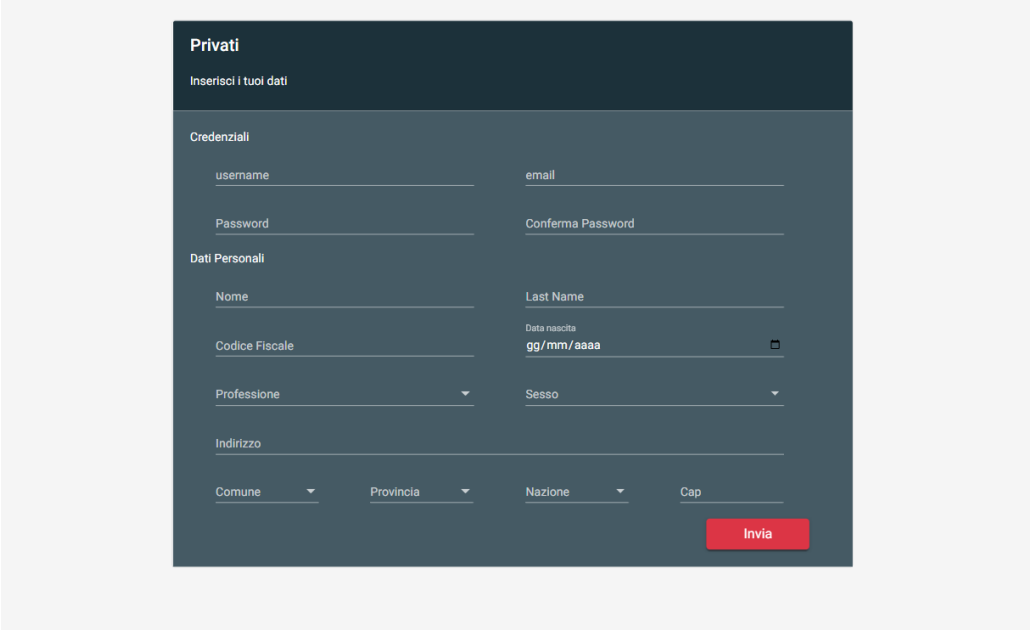


Figura 7.4: Registrazione cliente

Il cliente per effettuare la registrazione dovrà inserire i seguenti dati:

- username
- email

- password
- nome
- cognome
- codice fiscale
- professione
- sesso
- indirizzo
- comune
- provincia
- nazione
- cap

La scelta della professione avviene tramite un menù a tendina contenente una lista popolata mediante una chiamata di tipo http GET al microservizio DME Anagrafiche, in particolare all'end-point /anagrafiche/findAllProfessioni.

Alla pressione del bottone invio verrà, prima di tutto, effettuato un controllo del corretto inserimento dei dati, in seguito saranno effettuate una serie di chiamate in cascata per rendere effettiva la registrazione. La prima chiamata http sarà una chiamata di tipo POST al microservizio DME Security, in particolare all'endpoint /utente/createUser, il body di questa chiamata sarà di tipo RegisterUserDto:

```
class RegisterUserDto
{
    authUserDto: AuthUserDto;
    utentiDto: UtentiDto;
}

class AuthUserDto
{
    email: string;
    password: string;
    username: string;
}
class UtentiDto
{
    dtInserimento: Date;
    dtUltimaModifica: Date;
```

```
17                  dtUltimoAccesso: Date;
18                  codStatus: string;
19                  codProfilo: string;
20                  codTipologia: string;
21              }
22
```

Una volta ricevuta una response indicante il successo dell'operazione verrà inoltrata una seconda http request anche in questo caso di tipo POST, all'end-point /clienti/createCliente appartenente al microservizio DME Clienti. Il body della chiamata http conterrà un ClientDto al cui interno sono presenti tutti i campi della tabella `ana_Clienti` della figura **??** Se la response della chiamata dovesse indicare successo dell'operazione l'utente verrà reindirizzato alla pagina di login.

Registrazione Fornitore

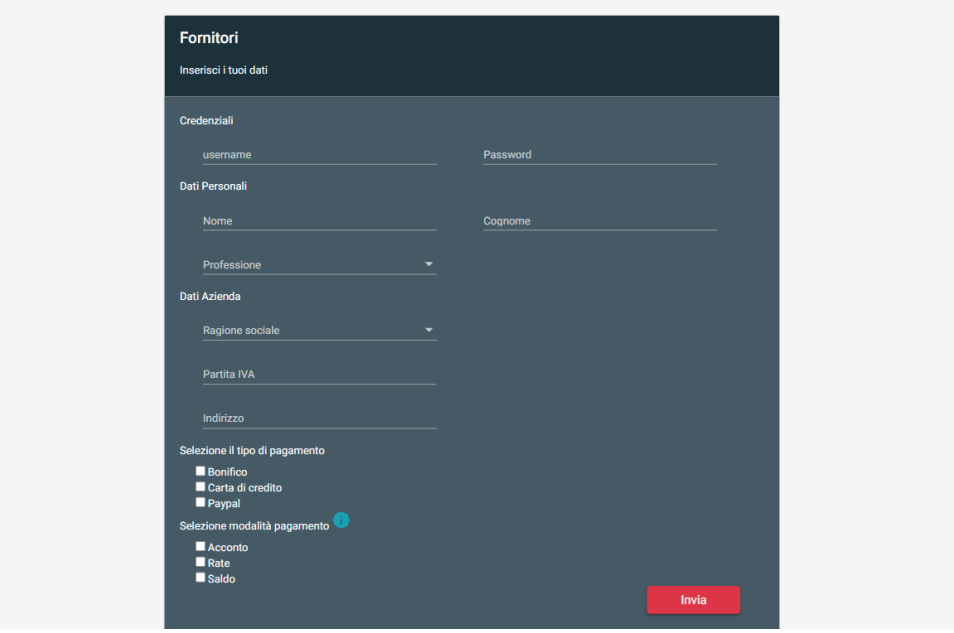


Figura 7.5: Registrazione Fornitore

Il fornitore per effettuare la registrazione dovrà inserire i seguenti dati:

- username
- password
- nome

- cognome
- professione
- Ragione sociale
- partita iva
- indirizzo
- tipo pagamento accettato
- modalità pagamento accettato

Le liste dei tipi di pagamento, delle modalità pagamento e delle professioni verranno popolate mediante 3 chiamate http GET al microserviio DME anagrafiche.

Il fornitore potrà scegliere i tipi di pagamento accettati mediante una lista di checkbox. Per quanto riguarda le modalità di pagamento, invece, sono possibili 3 alternative:

- Saldo: il pagamento avverrà in una sola rata.
- Acconto: il fornitore potrà scegliere una percentuale di acconto che il cliente dovrà pagare al momento dell'ordine e mentre la restante parte del totale dovrà essere versata entro la data di consegna del servizio.
- Rateizzazione: il fornitore potrà scegliere un numero di rate in cui il cliente potrà suddividere l'importo dovuto e ad ognuna di esse assegnare una percentuale.
  Nel caso in cui la somma delle percentuali delle rate non raggiunga il 100 sarà previsto un pagamento di saldo finale.
  Esempio: il fornitore sceglie come numero di rate 3 e assegna le seguenti percentuali Rata 1 = 10 Rata 2= 20 Rata 3= 30, il totale è 60 quindi il restante 40 verrà richiesto all'utente come pagamento finale.

Alla pressione del bottone "Invia", come nel caso della Registrazione Cliente, saranno effettuate una serie di chiamate in cascata. Nella chiamata al miscroservizio DME Security all'endpoint /utente/createUser, il body di questa chiamata sarà analogo a 7.2.1 Successivamente, dopo aver ricevuto una response adeguata, verrà effettuata una chiamata http POST al microservizio DME Fornitori all'end-point /fornitori/createFornitore. Il body di questa Request sarà di tipo FornitoriDto al cui interno sono presenti tutti i campi della tabella `ana_Fornitori` visionabile in figura 3.5

Infine, verranno effettuate le chiamate per comunicare la microservizio DME

Fornitori i tipi e le modalità di pagamento scelte dal fornitore, in particolare saranno 2 http Request di tipo POST

- **/tipoPagAccettato/createTipoPagAccettato**: il body sarà di tipo TipoPagAccettatoDtoReq contente tutti i campi della tabella `for_TipoPagAccettato` presente in 3.5
- **/modPagAccettato/createModPagAccettato**: il body sarà di tipo ModPagamentoAccettatoReq contente tutti i campi della tabella `for_ModPagAccettato` presente in 3.5

Ricevuto esito positivo da tutte le chiamate, il fornitore sarà reindirizzato alla pagina di login.

Login

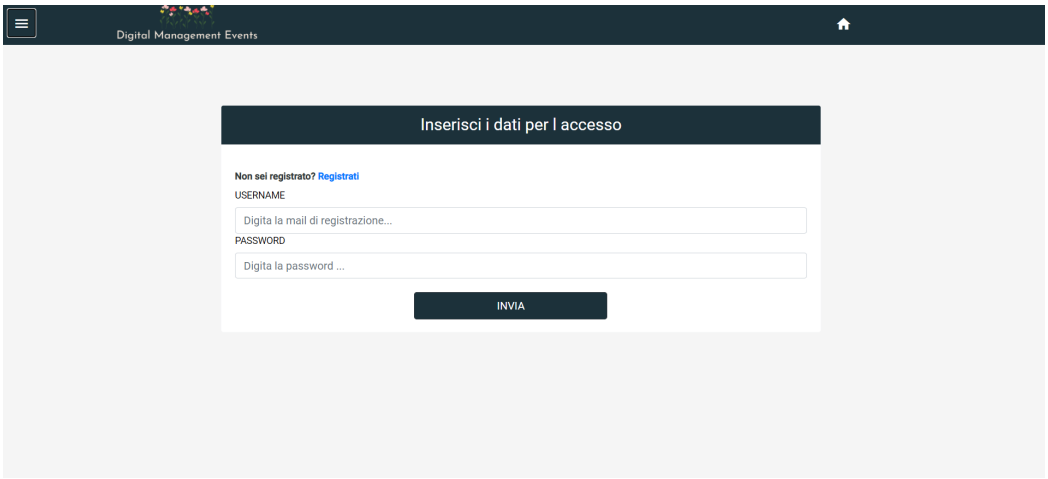


Figura 7.6: Login

Gli utenti registrati dovranno inserire le loro credenziali all'interno del form per poter accedere ai servizi della piattaforma. Premendo sul bottone invio verranno inoltrate delle chiamate ai microservizi di back end per verificare che l'utente sia effettivamente autorizzato ad accedere. La prima chiamata sarà di tipo POST al microservizio DME Security e sarà composta come segue:
`oauth/token?grant_type=password&username=utente &password=password`
Contiene un header Authorization con clientId e clientSecret, restituisce il jwtToken corrispondente all'utente e password inseriti nell'url. Ricevuta la response con esito positivo dell'operazione, si procederà alla richiesta dei dati dello User utilizzando lo userId attraverso la chiamata /utente/findUserById il cui body sarà composto come segue idUtente: userId . Superato il controllo del back end l'utente

verrà reindirizzato direttamente alla pagina privata rispettivamente del cliente o del fornitore in base alla tipologia di utenza scelta in fase di registrazione.

7.2.2 Navigazione Cliente

Pagina Privata

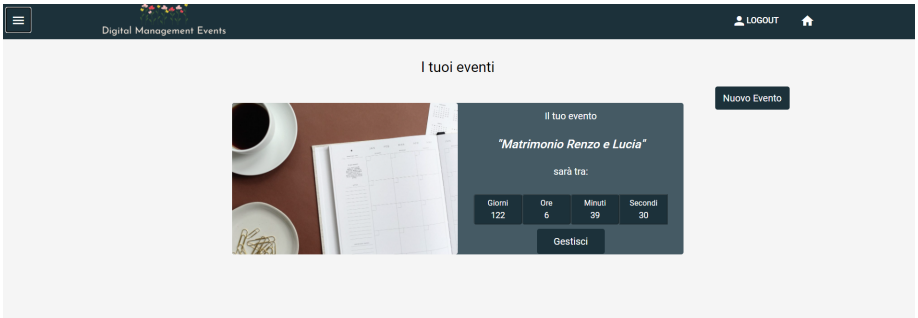


Figura 7.7: Pagina Privata

Il cliente in questa sezione potrà, sia vedere tutti gli eventi in fase di organizzazione tramite una lista di card, sia creare un nuovo evento facendo click sull'apposito bottone in alto a destra.

Ciascuna card presenta al suo interno il nome dell'evento, un count down che ricorda al cliente i giorni che mancano alla data dell'evento e un bottone per entrare nella gestione di quel particolare evento.

Le informazioni riguardanti gli eventi del cliente sono ottenute tramite una chiamata POST al microservizio DME Eventi, in particolare all'end point /eventi/-findEventiByIdCliente, il cui body sarà composto da idCliente: id. La response ottenuta sarà un vettore di EventoDto, ovvero un DTO contenente tutti campi della tabella `eve_Eventi` nella figura 3.7

Nuovo Evento

Per la creazione di un nuovo evento sarà prima di tutto necessario selezionare il tipo di Evento (Matrimonio, Compleanno,..) la cui lista è ottenuta mediante la chiamata di tipo GET /anagrafiche/findAllTipiEventi al microservizio DME anagrafiche.

Successivamente l'utente dovrà inserire i dati relativi all'evento

- Nome Evento
- Data Evento
- Provincia e Comune
- Budget Totale

- Indirizzo email secondo organizzatore nel caso fosse presente (ad esempio sposo/sposa, socio per eventi aziendali, ..)
- servizi desiderati, nella lista di checkbox laterale troviamo un insieme di servizi tra i quali il cliente può scegliere per il proprio evento.

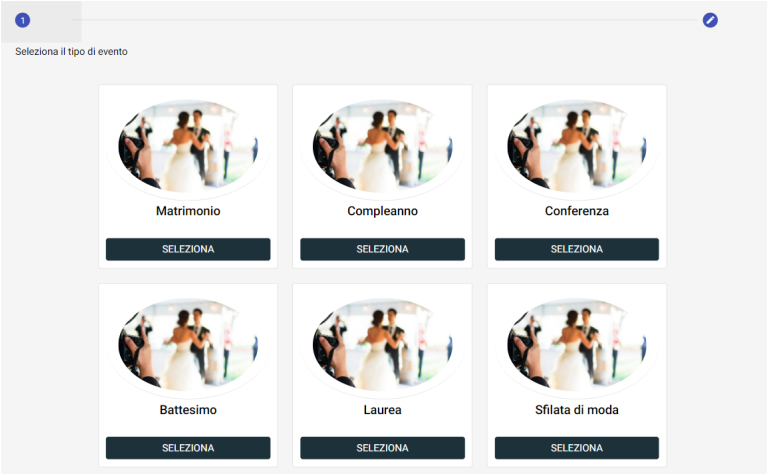


Figura 7.8: Selezione Tipo Evento

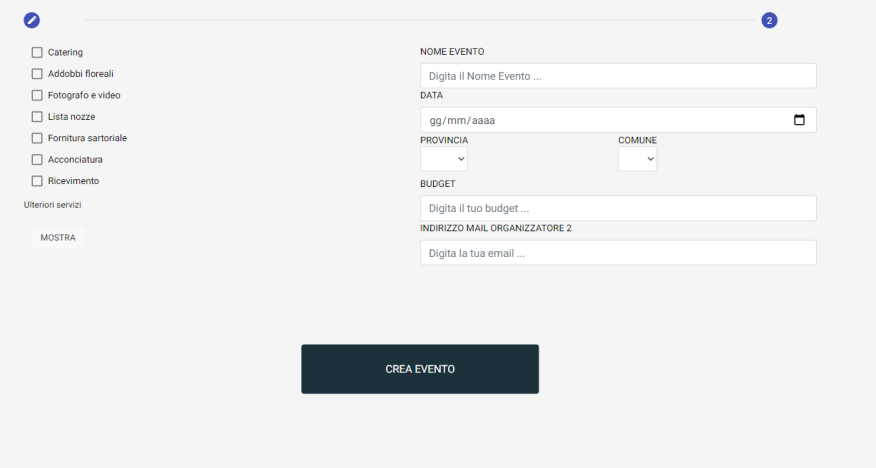


Figura 7.9: Creazione Evento

Una volta inseriti tutti questi campi il cliente proseguirà premendo il bottone "CREA EVENTO" che scatenerà una serie di chiamate al microservizio DME Eventi. La prima chiamata sarà una http POST all'end-point /eventi/createEvento in cui il body è di tipo EventoDto.

Dopo aver ricevuto una response positiva alla richiesta precedente verrà effettuata una chiamata per ciascuna Categoria di Servizio scelta all'end-point /eventoServizi/createEventoServizi del microservizio DME Eventi. Il body di ciascuna chiamata sarà un EventoServiziDto contente gli stessi campi della tabella `eve_Evento_Servizi` presente in 3.7. Terminate tutte le chiamate l'utente verrà reindirizzato alla pagina di gestione evento

Gestione Evento

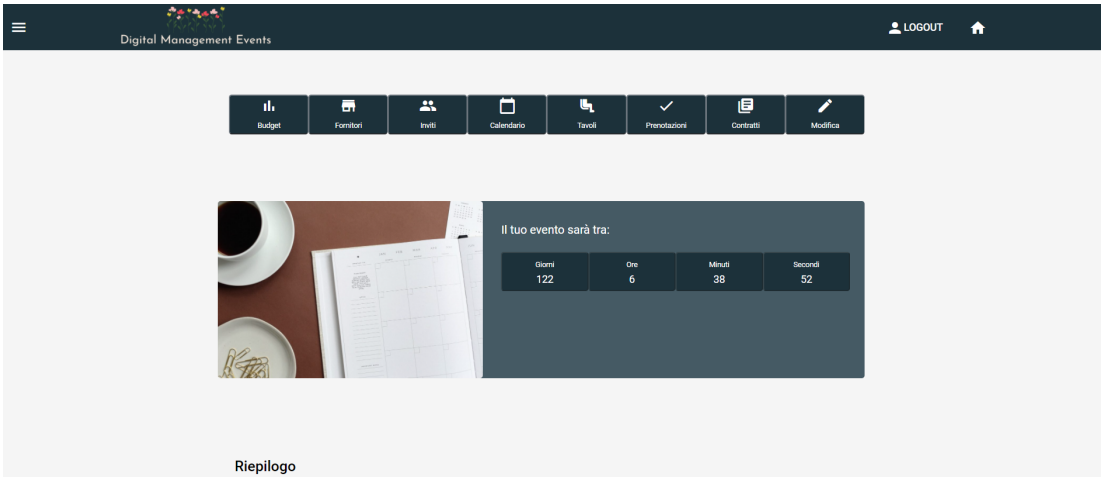


Figura 7.10: Gestione Evento

Questa schermata rappresenta l'homepage del particolare evento, da qui l'utente potrà accedere alle varie sezioni per la gestione dell'evento tramite la barra di navigazione. In particolare le sezioni raggiungibili dal cliente tramite questa pagina saranno:

- Budget: permetterà al cliente di definire un budget per ogni servizio selezionato in fase di creazione dell'evento.
- Fornitori: attraverso questa sezione potranno essere visualizzati i vari fornitori raggruppati per categoria.
- Inviti: l'organizzatore potrà stilare una lista di invitati e gestire ciascun invitato.
- Tavoli: in questa sezione il cliente potrà scegliere come posizionare gli invitati durante l'evento.
- Agenda: permetterà di avere sotto controllo tutti gli appuntamenti presi.

- Ordini: il cliente potrà visualizzare e gestire gli ordini effettuati.
- Contratti.
- Modifica Evento: in questa sezione sarà possibile modificare i dati dell'evento.

Budget

Utilizzando questo tool l'utente potrà definire un budget per ogni Categoria Servizio selezionata in fase di creazione evento e potrà controllare se la Stima Attuale, ottenuta attraverso la somma dei budget specifici, rientra nella budget totale definito in fase di creazione dell'evento. Per rendere i valori inseriti persistenti verrà

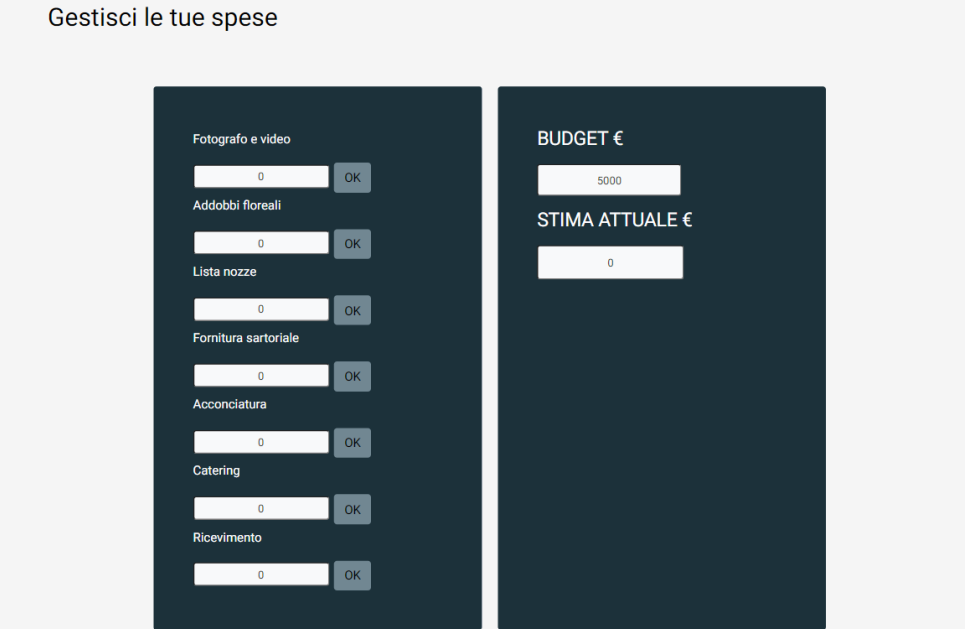


Figura 7.11: Gestione Budget

effettuata una chiamata alla pressione del bottone "OK" accanto ad ogni categoria di servizio selezionata. La chiamata http sarà di tipo PUT verrà inoltrata all' end point /eventoServizi/updateEventoServizi del microservizio DME Eventi, il body sarà di tipo EventoServiziDto

Modifica Evento

In questa sezione potranno essere modificati i dati dell'evento come nome, data, luogo e categorie servizio selezionate.

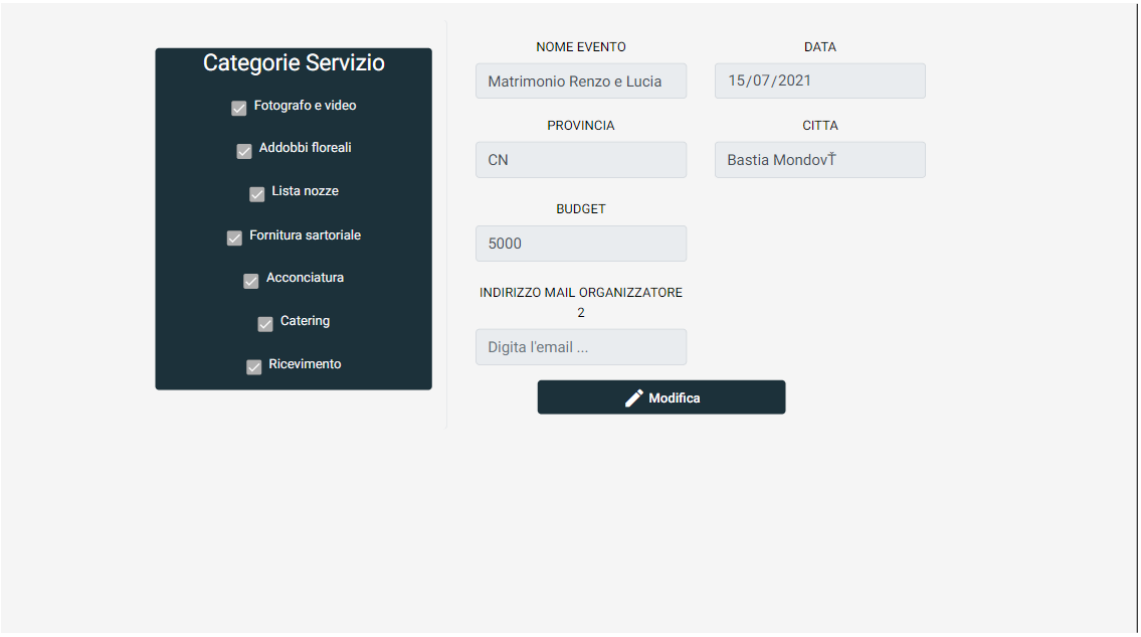


Figura 7.12: Modifica Evento

Fornitori

Il cliente accedendo a questa pagina visualizzerà la lista dei fornitori adatti alle sue necessità. La lista dei fornitori è ottenuta mediante una serie di chiamate http GET al miscroservizio DME Vetrine. Verrà effettuata una chiamata per ogni categoria di servizio selezionata dall'utente per il suo evento all'endpoin fornitoriVetrine/findFornitoriVetrinaByCodCategoriaServizio. Dalle response delle varie chiamate si otterrà un vettore composto da elementi di tipo VetrinaDto contente tutti i campi della tabella `vet_Fornitori_Vetrine` consultabile nella figura 3.9.

Questa schermata permetterà al cliente di trovare il fornitore più adatto alle proprie esigenze. L'utente sarà aiutato nella ricerca grazie ai filtri per categoria servizio. I filtri per recensioni e data disponibile non sono invece stati implementati.

La schermata di dettaglio di ciascun fornitore sarà visionabile facendo click su bottone "Vedi Dettagli" che porterà l'utente alla consultazione della vetrina del fornitore selezionato

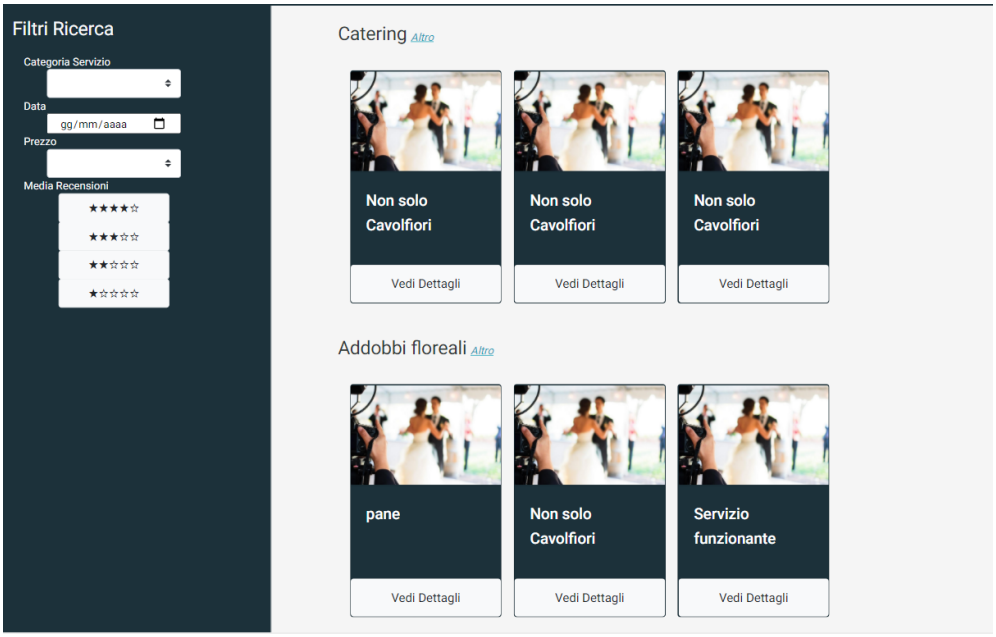


Figura 7.13: Lista Fornitori

Vetrina

La vetrina fornisce una serie di informazioni utili al cliente come la lista dei prodotti con il relativo prezzo, gli orari di apertura, i tipi e le modalità di pagamento accettati. In questa sezione l'utente potrà procedere con la richiesta di appuntamento o all'inoltro di un ordine.

La richiesta di Appuntamento può essere compilata completando l'apposito Form nella figura 7.15 inserendo la Data dell'appuntamento, l'orario ed eventuali note per il fornitore. Questa operazione sarà finalizzata con l'invio di una sequenza di http request di tipo POST al microservizio DME Agenda.

- Prima di tutto verrà creato l'appuntamento contattando l'endpoint /appuntamento/createAppuntamento il body della request sarà un AgendaDto contenente al suo interno tutti i campi della tabella `age_Appuntamento` presente nella figura 3.10.

- In seguito verrà associato l'appuntamento sia al cliente che al fornitore interessati mediante le rispettive chiamate all'endpoint /clienteAppuntamento/createClienteAppuntamento. Il body della request sarà un AgendaDto contenente al suo interno tutti i campi della tabella `age_Cliente _Appuntamento` presente nella figura 3.10, e all'endpoint /fornitoreAppuntamento/createFornitoreAppuntamento, il body della request sarà un AgendaDto contenente al suo interno

tutti i campi della tabella `age_Fornitore_Appuntamento` presente nella figura 3.10.

L'inoltro dell'ordine avviene completando il form in figura 7.16 quindi selezionando i prodotti a cui si è interessati, il tipo e la modalità di pagamento, la data e l'orario in cui si vuole quel determinato prodotto o servizio. Questa operazione sarà finalizzata con l'invio di una http request di tipo POST all'end point /ordini/createOrdine del microservizio DME Clienti. Il body sarà un OrdiniDto contenente tutti i campi della tabella `cli_Ordini` presente nella figura 3.4.

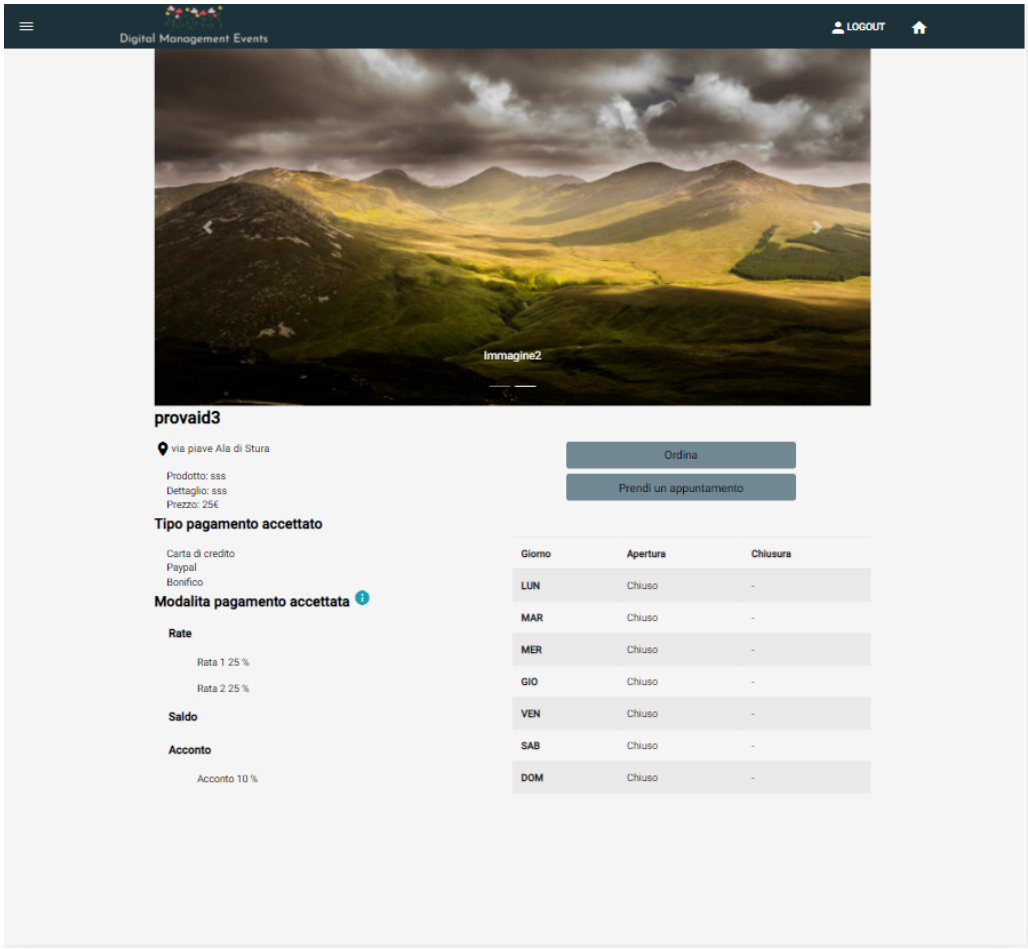


Figura 7.14: Vetrina

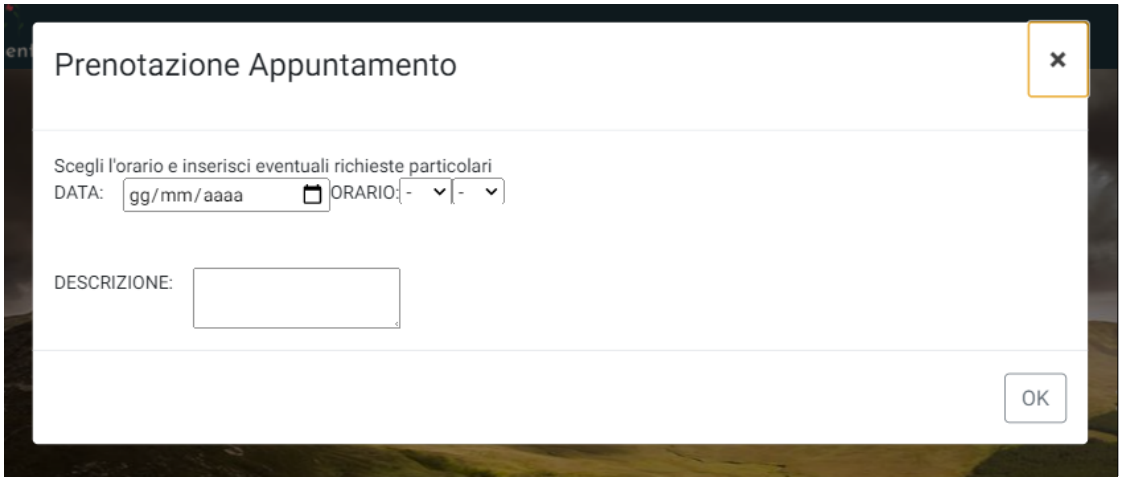


Figura 7.15: Appuntamento

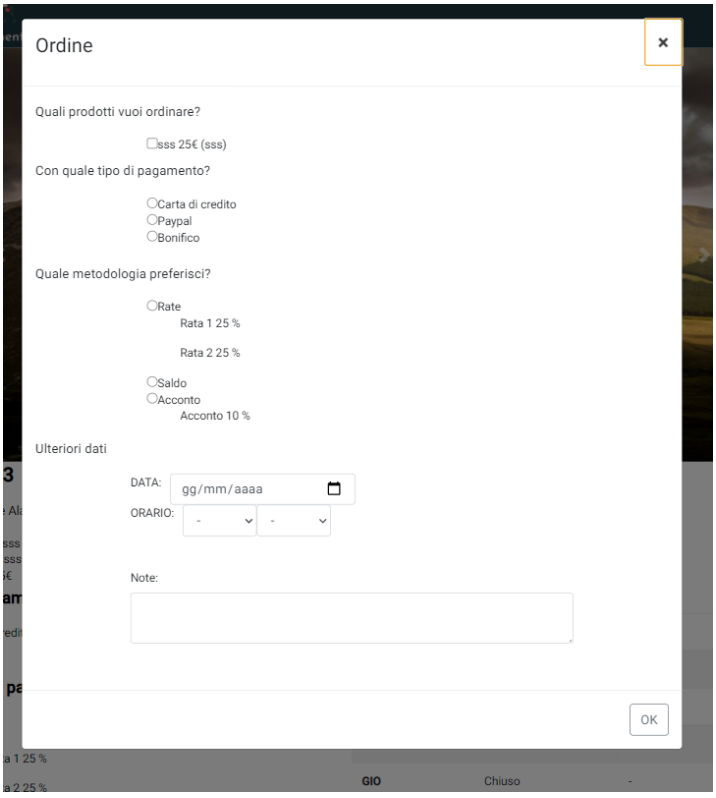


Figura 7.16: Ordine

Agenda Appuntamento

In questa sezione l'utente potrà gestire i suoi appuntamenti. Sulla sinistra vediamo un componente con 3 tab in cui cono filtrati gli appuntamenti per stato:

- Confermati.
- In Sospeso: il fornitore ha modificato data o orario dell'appuntamento dunque il cliente potrà scegliere se accettare, proporre un altro orario/data mediante il tasto modifica o cancellare l'appuntamento.
- In attesa di conferma: la richiesta è stata inviata da parte del cliente ma non è stata ancora accettata dal fornitore.

Sulla destra invece è presente un calendario per visualizzare gli appuntamenti. In questo caso per facilitare la consultazione sono stati inseriti dei pallini verdi, rossi e gialli ad indicare rispettivamente gli appuntamenti confermati, in attesa di conferma e in sospeso.

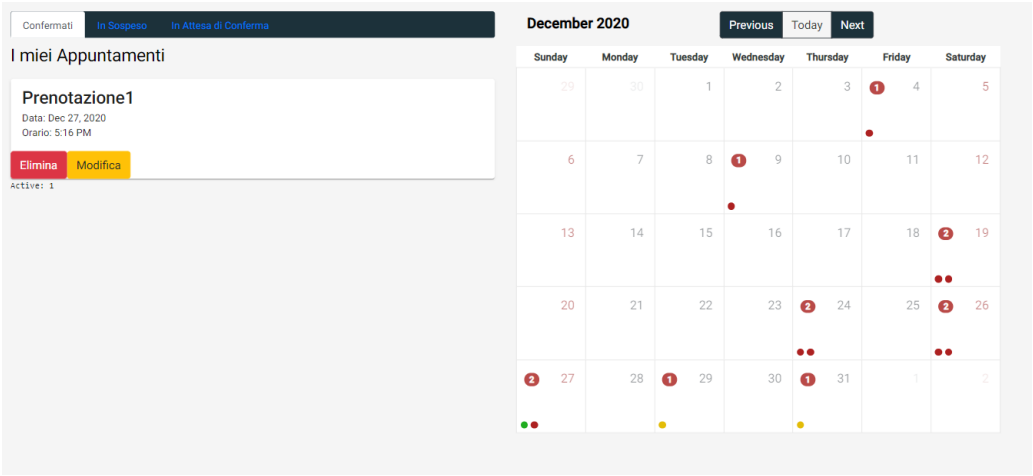


Figura 7.17: Agenda Appuntamento

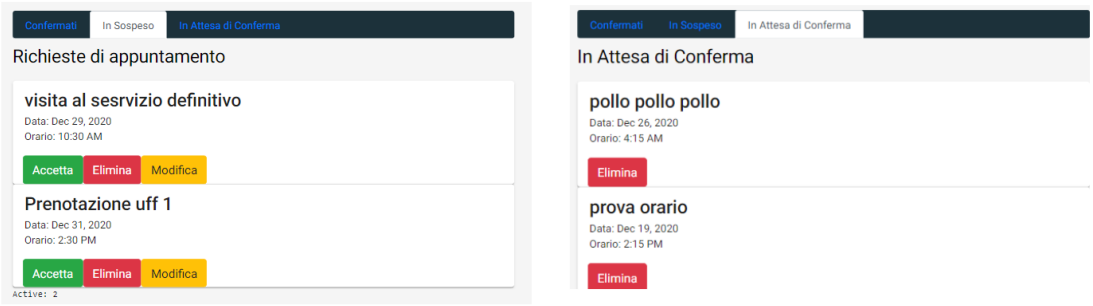


Figura 7.18: Stato Appuntamenti

Le operazioni di modifica e conferma verranno effettuate attraverso una chiamata http PUT all'endpoint /appuntamento/updateAppuntamento del microservizio DME Agenda, mentre l'operazione di cancellazione verrà effettuata attraverso una chiamata http DELETE all'endpoint /appuntamento/deleteAppuntamentoByIdAppuntamento del microservizio DME Agenda.

Inviti

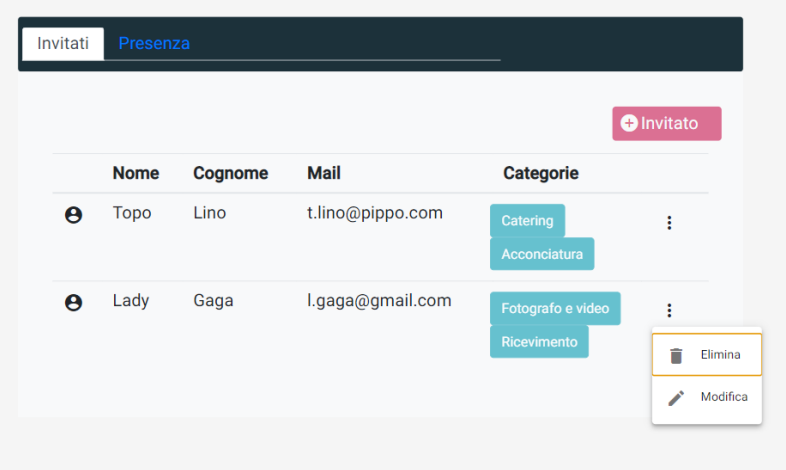


Figura 7.19: Lista invitati

In questa sezione l'organizzatore potrà gestire i propri invitati. Mediante il tasto in alto a destra il cliente potrà aggiungere un invitato inserendo nome, cognome, email e selezionando a quali momenti dell'evento quella persona è invitata. Nella tabella ci sarà un elenco di tutti gli invitati con tasto per ciascun invitato che permetterà di cancellare l'invito o modificarlo. La tabella viene popolata mediante una sequenza di http request al microservizio DME Eventi:

- Verrà contattato l'end-point/relServizioInvitati /findRelServizioInvitatiByIdEventoServizio per ogni Categoria di Servizio scelta per l'evento, il body sarà idEventoServizio: id
- Per ogni Categoria di Servizio verrà ricevuto un vettore con l'idInvitato dunque per ogni invitato all'interno del vettore sarà effettuata una chiamata http POST all' end-point /invitati/findInvitatiByIdInvitato il cui body sarà idInvitato: id. Il risultato di questa operazione sarà un insieme di vettori con elementi del tipo InvitatoDto contenente gli stessi campi della tabella `eve_Invitati` presente nella figura 3.7.

Ordini

In questa pagina l'utente potrà vedere gli ordini effettuati, consultare lo stato e i dettagli. Il link 'Dettaglio vetrina' indirizzerà l'utente alla vetrina del Fornitore da cui ha effettuato l'ordine mentre il link 'Dettagli ordine' aprirà una finestra in cui verranno visionati i prodotti ordinati con la relativa quantità e prezzo.

Nel caso l'ordine sia stato accettato, il cliente in questa sezione potrà pagare secondo il metodo di pagamento scelto durante l' ordine. Per ogni pagamento verrà contattata la chaincode in modo da registrare la transazione.

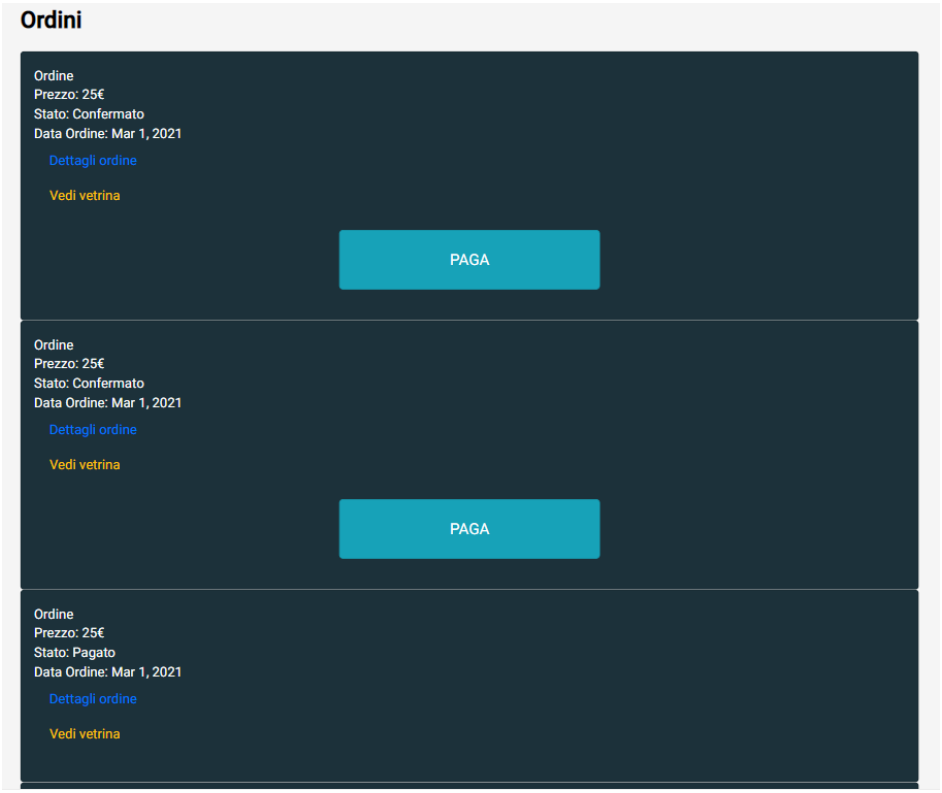
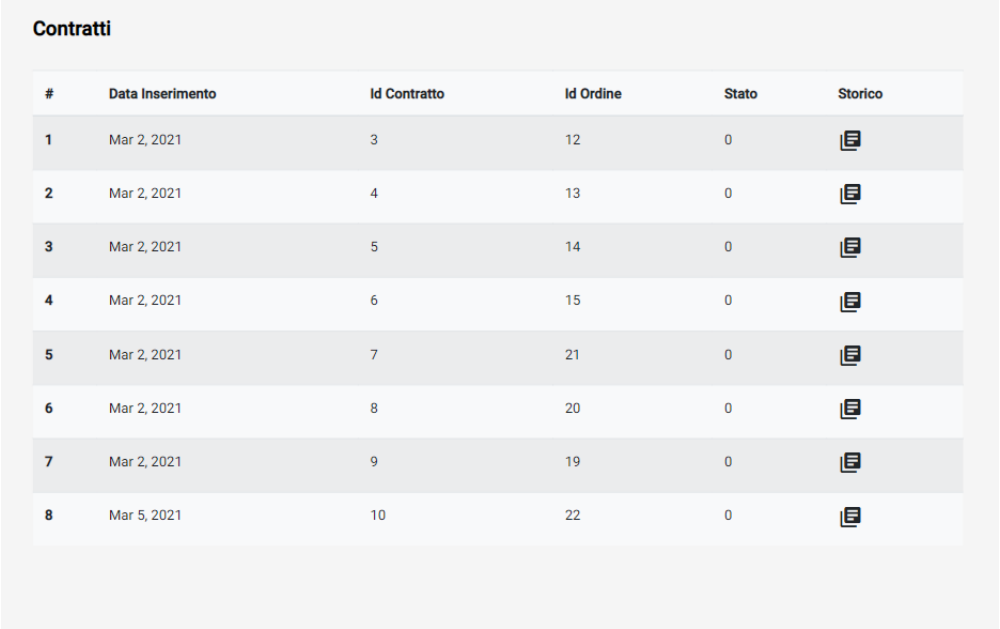


Figura 7.20: Lista Ordini

Contratti

La sezione Contratti presenta la lista degli smartcontract dell'utente per ognuno di essi è possibile consultare lo storico delle transazioni effettuate premendo l'apposito bottone. Al click del bottone verrà effettuata una richiesta alla blockchain per ottenere la lista delle transazioni.



Contratti

#	Data Inserimento	Id Contratto	Id Ordine	Stato	Storico
1	Mar 2, 2021	3	12	0	
2	Mar 2, 2021	4	13	0	
3	Mar 2, 2021	5	14	0	
4	Mar 2, 2021	6	15	0	
5	Mar 2, 2021	7	21	0	
6	Mar 2, 2021	8	20	0	
7	Mar 2, 2021	9	19	0	
8	Mar 5, 2021	10	22	0	

Figura 7.21: Lista Contratti

Storia transazioni

#	Data Contratto	ID Evento	ID Vetrina	ID Fornitore	Prezzo	Pagato	Stato
1	Mar 1, 2021	45	90123	90049	25	6	PAGAMENTO RATA

Figura 7.22: Storico Transazioni

7.2.3 Nav Fornitore

Pagina Privata Fornitore

Accedendo alla propria pagina privata, il fornitore visionerà i servizi offerti e potrà scegliere se crearne uno nuovo con il tasto in alto a destra oppure gestire un servizio già esistente con l'apposito bottone nelle card. Nella gestione del servizio il fornitore visualizzerà una pagina identica alla Vetrina Visualizzata dal fornitore ma invece che i tasti "Prendi Appuntamento" e "Ordina" ci saranno "Modifica Servizio", "Gestione Ordini" e "Gestione Appuntamenti". La gestione degli appuntamenti e degli ordini sono analoghe a quelle del cliente con la sola eccezione che nella sezione ordini il fornitore non avrà il tasto 'PAGA' e potrà scegliere se confermare l'ordine.

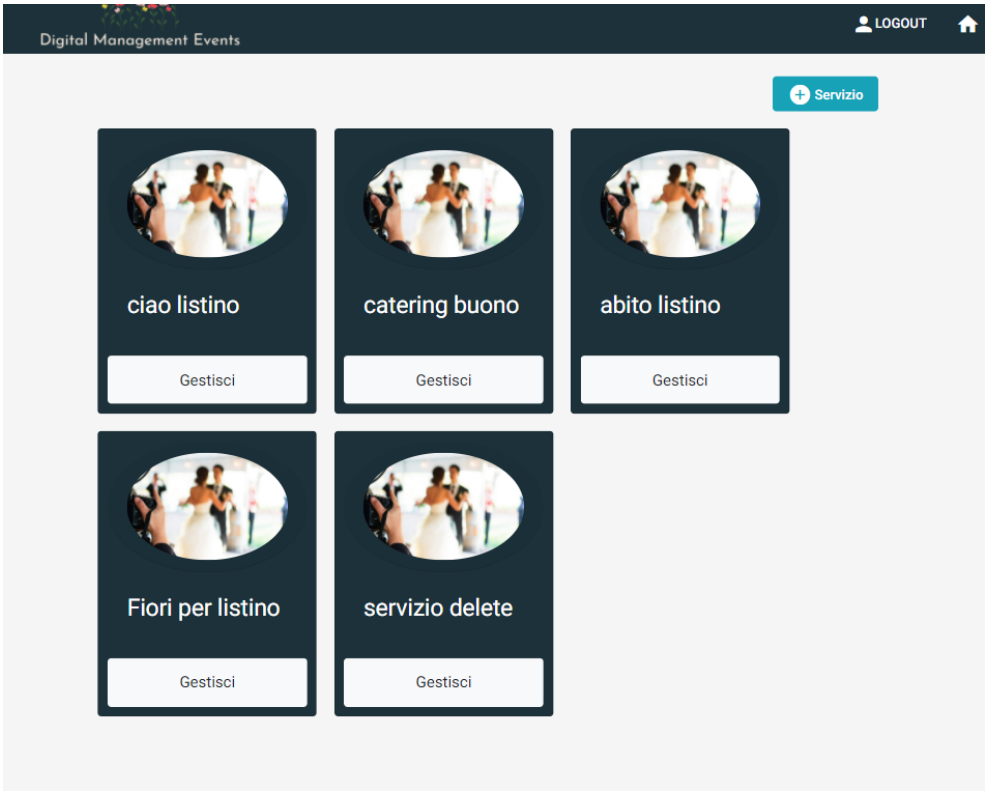


Figura 7.23: Lista Servizi

Nuovo Servizio

Per aggiungere un nuovo servizio il fornitore dovrà inserire i seguenti dati:

- Nome del Servizio
- Nazione, Provincia, Comune, Indirizzo
- Categoria di Servizio
- Prodotti con i relativi dettagli (Nome Prodotto, Descrizione, Prezzo, Descrizione Prezzo)
- Orari di Apertura del punto Vendita

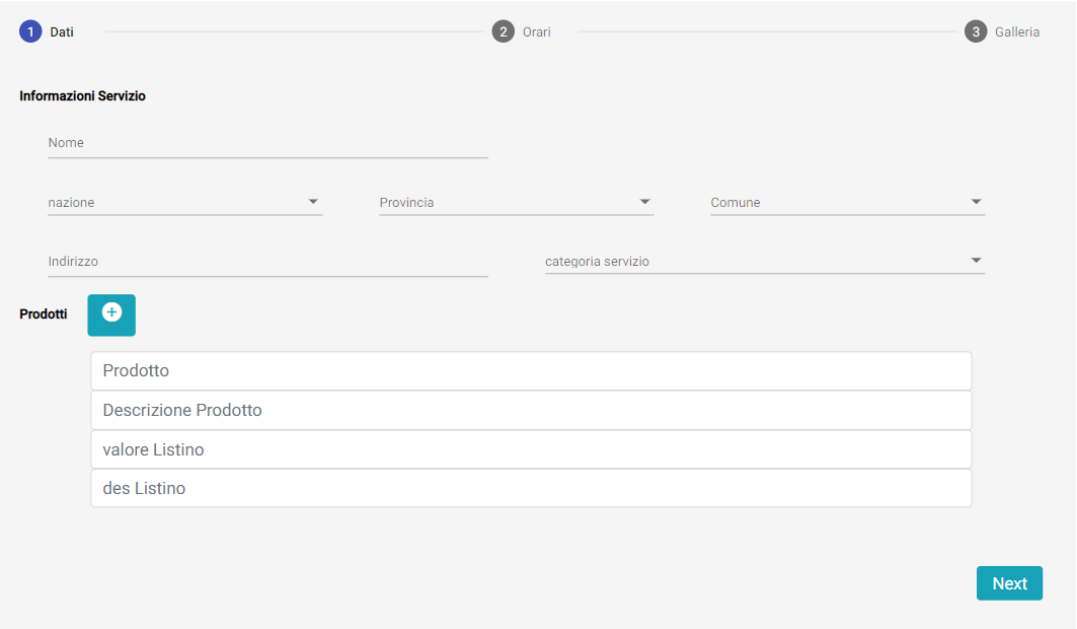


Figura 7.24: Nuovo Servizio Dati

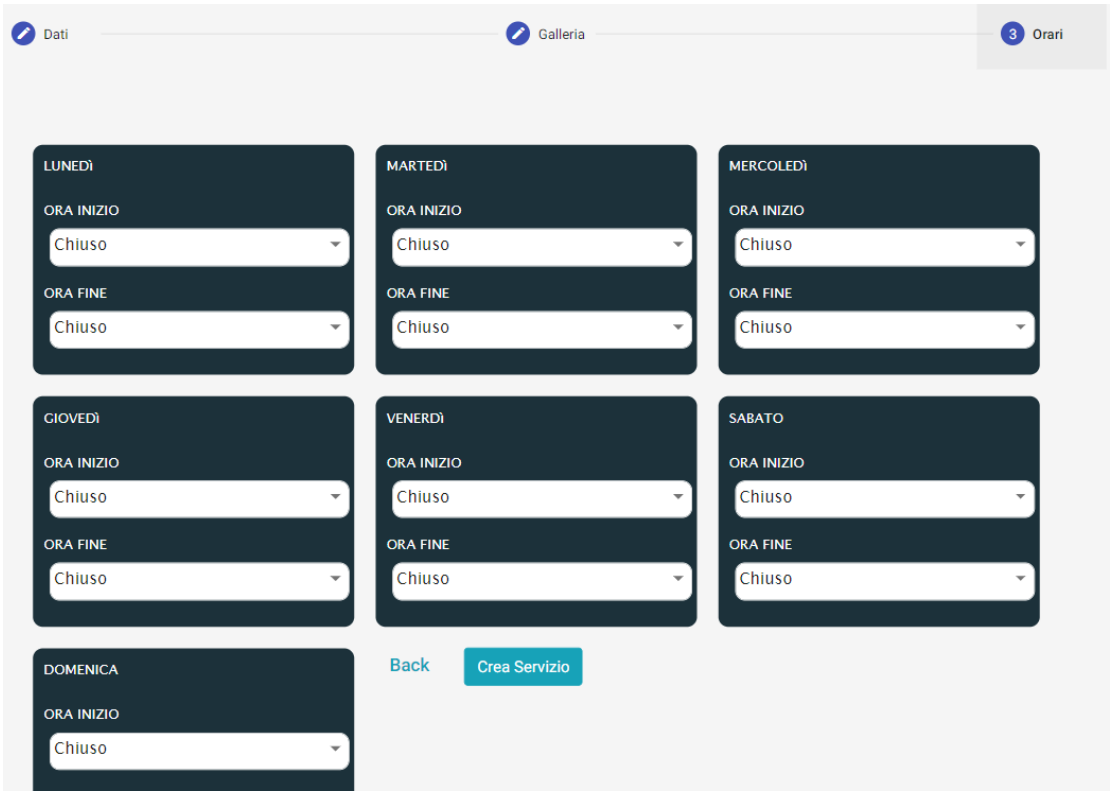


Figura 7.25: Nuovo Servizio Orari

Modifica Servizio

In questa sezione il fornitore potrà modificare le informazioni del servizio selezionato: nome, orari, indirizzo, prodotti.

Capitolo 8

Conclusioni

8.1 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.

8.2 Sviluppi futuri

A seguito del lavoro svolto fino 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.[2ex] 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.

8.3 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] angular.io. *Introduction to Angular concepts.* [Online; in data 24-febbraio-2021]. 2021. URL: https://angular.io/guide/architecture (cit. alle pp. 54, 55).

[4] Gavin Bierman, Martín Abadi e Mads Torgersen. «Understanding TypeScript». In: *ECOOP 2014 – Object-Oriented Programming.* A cura di Richard Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 257–281 (cit. a p. 56).

[5] TypeScript. *TypeScript for the New Programmer.* [Online; in data 24-febbraio-2021]. 2021. URL: https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html (cit. a p. 56).

[6] RxJS. *Introduction.* [Online; in data 24-marzo-2021]. 2021. URL: https://rxjs-dev.firebaseapp.com/guide/overview (cit. a p. 56).

[7] Fabio Biondi. *Introduzione RXJS.* [Online; in data 18-marzo-2021]. 2021. URL: https://training.fabiobiondi.io/2019/09/15/introduzione-rxjs-parte-1-fundamentals/ (cit. a p. 57).

[8] Michael Jakl. *REST Representational State Transfer* (cit. a p. 58).

[9] Architectural Styles e the Design of Network-based Software Architectures. «A systematic literature review of test breakage prevention and repair techniques». In: (2000) (cit. a p. 58).

[10] Wikipedia. *Representational State Transfer — Wikipedia, L'enciclopedia libera.* [Online; in data 24-febbraio-2021]. 2021. URL: http://it.wikipedia.org/w/index.php?title=Representational_State_Transfer&oldid=118821012 (cit. a p. 58).

[11] M. Jazayeri. «Some Trends in Web Application Development». In: *Future of Software Engineering (FOSE '07)*. 2007, pp. 199–213. DOI: 10.1109/FOSE.2007.26 (cit. a p. 59).

[12] Microsoft. *Scegliere tra app Web tradizionali e a pagina singola*. [Online; in data 24-febbraio-2021]. 2021. URL: https://docs.microsoft.com/it-it/dotnet/architecture/modern-web-apps-azure/choose-between-traditional-web-and-single-page-apps (cit. a p. 60).

[13] Microsoft. *Creazione dell'interfaccia utente composita basata su microservizi*. [Online; in data 13-gennaio-2021]. 2021. URL: https://docs.microsoft.com/it-it/dotnet/architecture/microservices/architect-microservice-container-applications/microservice-based-composite-ui-shape-layout (cit. a p. 61).

[14] Michael Nebeling e Moira C. Norrie. «Responsive Design and Development: Methods, Technologies and Current Issues». In: *Web Engineering*. A cura di Florian Daniel, Peter Dolog e Qing Li. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013 (cit. a p. 62).