ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN SCIENZE E TECNOLOGIE INFORMATICHE

# PROGETTAZIONE DI UN SISTEMA DI PRENOTAZIONE ON LINE CON DBMS NOSQL

Relazione finale in
Laboratorio di Basi di Dati

Relatore
Prof. Matteo Golfarelli

Presentata da
Gianni Bernabè

Sessione III
Anno Accademico 2013/2014

# Indice

# Introduzione

L'industria del turismo ha subito profondi cambiamenti negli ultimi anni. Il motore principale di questo cambiamento è stata la tecnologia. Connessioni ad Internet sempre più veloci e accessibili insieme all'avvento di dispositivi come smartphone e tablet hanno radicalmente cambiato il modo in cui le persone pianificano e prenotato i propri viaggi.

Da questo cambiamento è nata un' industria parallela a quella tradizionale: la "online travel industry". Questo nuovo mercato è stato subito popolato da aziende già attive nel mercato tradizionale che si sono reinventate per approfittare dei tempi che cambiavano. Al giorno d'oggi sono presenti, oltre alle suddette, anche imprese specializzate nel solo mercato online dei viaggi.

A marzo 2013 [Etc13], 183 milioni di persone hanno visitato almeno un sito online di viaggi. Sempre nel 2013 il 58% degli europei ha prenotato le proprie trasferte via Internet, contro il 13% di dieci anni fa. La diffusione di dispositivi portatili connessi in rete ha dato un grossa spinta a questo mercato. A fine 2014 il 15% delle prenotazioni online è stata fatta da uno smartphone  o un tablet. Per il 2015 si prevede di arrivare fino ad un quinto del totale.

È chiaro che questa industria è tuttora in grande crescita e il suo picco massimo sembra ancora lontano. EasyMarket è un azienda attiva in questo settore e questa tesi tratterà di un sistema sviluppato per suo conto. Nello specifico saranno descritte la progettazione, lo sviluppo e la messa in

produzione di un modulo di business intelligence. Il suo scopo sarà fornire la piattaforma più adatta per attuare piani di sviluppo aziendali e campagne commerciali. In primo luogo il lettore verrà messo a conoscenza del contesto in cui il sistema dovrà lavorare. Sarà descritta l'azienda e il suo ruolo nel settore del turismo online. Verranno elencate le problematiche principali che questo settore pone e come EasyMarket le abbia risolte. Verrà mostrata, in termini generali, l'architettura e di quali strumenti si avvale per portare avanti i propri progetti. Il secondo capitolo fornirà una breve panoramica sul mondo dei database non relazionali. Indicherà su quali basi concettuali essi si poggiano e quali vantaggi (e svantaggi) portano con loro. Nel terzo capitolo saranno elencate le principali tecnologie che il sistema oggetto di questa tesi utilizza. In particolare verranno mostrate le principali funzionalità di MongoDB e le modalità di accesso ai dati in esso contenuti. Infine nell'ultimo capitolo il lettore avrà visione di tutto il ciclo di vita del nuovo componente. Oltre alle specifiche richieste, saranno mostrate tutte le fasi di sviluppo, dall'analisi ai test finali.

# Capitolo 1

# Sistemi di prenotazione online

Questo capitolo si compone di due parti. La prima è una panoramica sull'industria turistica, in particolar modo di quella online. Poi saranno trattate le problematiche che un'azienda del settore deve necessariamente affrontare. Si parlerà delle relazioni con i fornitori e i clienti, nonché degli aspetti legali del caso. Nella seconda parte verrà introdotta EasyMarket, spiegando come gestisce il suo business e come ha risolto i problemi elencati nella prima parte. Si passerà poi a illustrare la sua architettura informatica, partendo dalle criticità che gestisce. Saranno elencati i principali componenti nonché le tecnologie maggiormente sfruttate.

## 1.1 Turismo web-based

Gli ultimi anni hanno profondamente cambiato le modalità con cui le persone pianificano e prenotano le proprie vacanze. Per andare incontro a questo nuovo mercato le aziende del settore hanno costruito dei sistemi di prenotazione online. Questi permettono a chiunque di cercare e acquistare la propria vacanza in autonomia. All'inizio questi “motori di ricerca” si limitavano a presentare prodotti che rispondevano a certi criteri di ricerca. Col tempo però, le aziende hanno cominciato ad usarli per mettere in atto campagne di marketing per massimizzare i profitti.

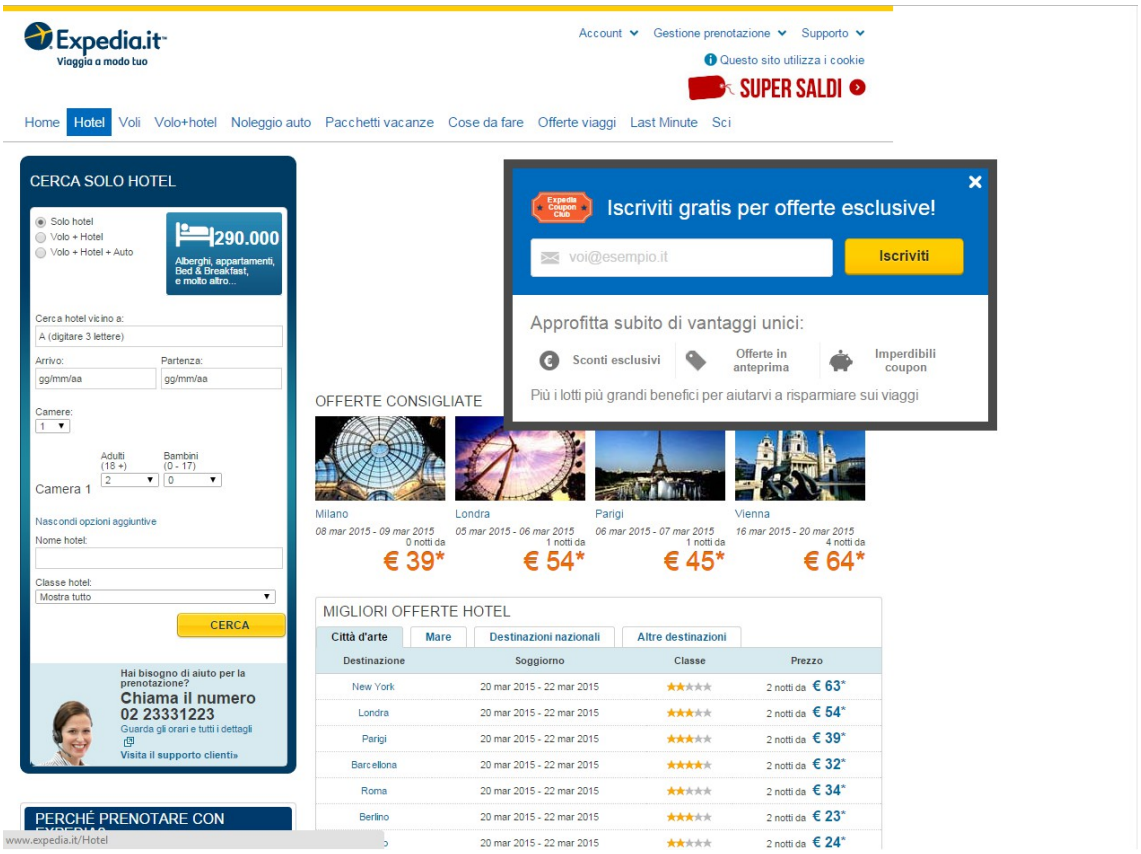


*Figura 1.1 : Interfaccia di ricerca hotel di Expedia.it*

Come si può vedere in figura 1.1, le aziende hanno trasformato i propri motori in vere e proprie vetrine. Lo scopo è quello di spingere il visitatore a scegliere offerte più vantaggiose per il gestore del servizio.

Per costruire un sistema di questo tipo è necessario risolvere alcune questioni. In primis, reperire il prodotto. I fornitori di questo mercato offrono, a chi lo richieda, la possibilità di interrogare i loro servizi. Si tratta solitamente di servizi web progettati non per utenti umani, ma per applicazioni esterne ad essi. Per poterli interrogare è necessario stipulare un contratto di fornitura con l'azienda che li gestisce. Un altro aspetto che deve essere assolutamente considerato, è il cliente. La prenotazione di un servizio turistico è a tutti gli effetti un contratto legale. Azienda e cliente sono vincolati a rispettarlo in tutti i suoi termini. È quindi necessario

conoscere tutte le implicazioni legali che questi tipi di contratti comportano. Infine bisogna prevedere le modalità con cui il cliente può pagare i propri acquisti. Devono essere semplici, sicure e affidabili agli occhi dell'acquirente.

## 1.2 EasyMarket

EasyMarket S.p.a. [Eas15] è una società del gruppo TUI Travel, primo operatore turistico mondiale. È una società multi-fornitore e multi-prodotto, il che significa che offre diversi prodotti, prelevati da molteplici fornitori. I suoi pacchetti prevedono hotel, voli, escursioni, trasferimenti e autonoleggi. I suoi interessi commerciali comprendono sia il mercato B2B che B2C, quindi sia privati che altre società. I clienti possono visionare i prodotti EasyMarket attraverso i suoi molteplici front-end (figura 1.2).


*Figura 1.2 : Esempi di domini gestiti da EasyMarket. da sinistra: tui.it, lol.travel, revolution.travel.*

Per quanto riguarda il B2B, EasyMarket offre i propri servizi alle agenzie di viaggio. Attraverso l'applicazione Revolution[1] (figura 1.3), le agenzie accreditate possono prenotare pacchetti turistici per conto dei loro clienti.

EasyMarket possiede un call center dedicato al servizio clienti. I suoi dipendenti lavorano per risolvere i problemi che questo tipo di azienda deve affrontare.

1 Revolution, home page - http://www.revolution.travel

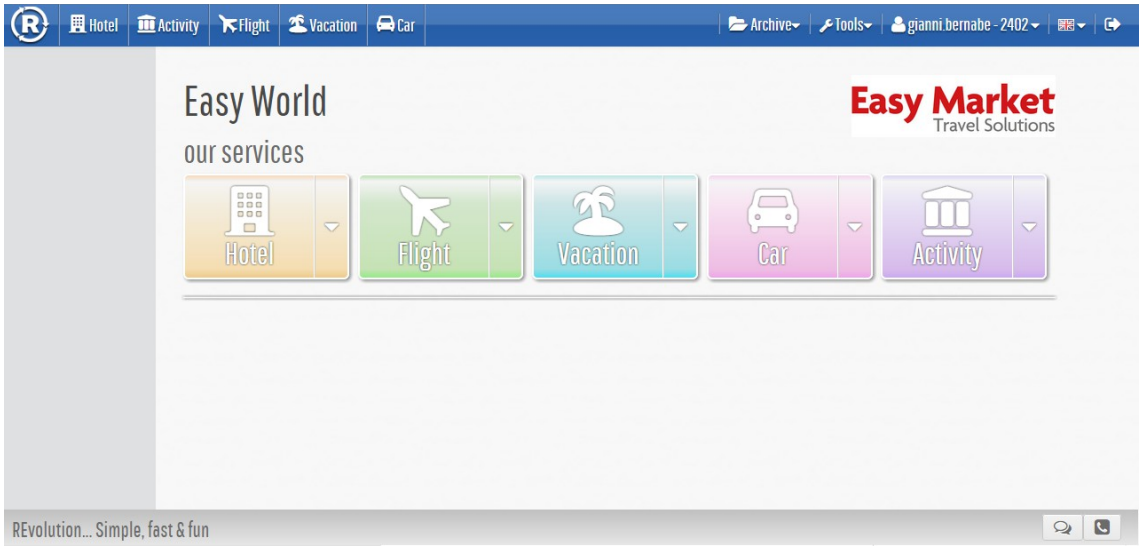


*Figura 1.3 : La pagina principale di Revolution*

**Gestione pratiche:** le prenotazioni fatte, specialmente quelle del settore B2C, non possono essere gestite completamente in automatico. Un pratica può essere accettata solo dopo alcuni controlli sul cliente e sulla pratica stessa. Questo processo impedisce che prenotazioni fasulle o fraudolente vengano passate ai fornitori. In conclusione EasyMarket cerca di assicurarsi di ricevere il pagamento dell'acquirente, prima di pagare a sua volta il fornitore. Per il settore B2B questo processo è meno rigoroso. Le agenzie interessate a collaborare con l'azienda possono richiedere la stipula di un contratto. Dopo aver effettuato tutti i controlli del caso l'agenzia viene approvata e gli vengono fornite le credenziali d'accesso dell'applicazione B2B (Revolution). L'attività all'interno di Revolution è scrupolosamente tracciata, per cui controllare ogni singola pratica, in questo caso, non è necessario.

**Customer care:** può succedere che un cliente riscontri un problema nell'utilizzo del sistema EasyMarket. Il call center si occupa di fornire assistenza in questi casi. Tra le richieste gestite figurano problematiche correlate alle prenotazioni e attivazioni di nuove agenzie.

**Controllo qualità:** durante lo svolgimento delle loro mansioni, gli operatori possono rilevare bug di programmazione e/o incongruenze nei dati. Questi vengono notificati al reparto competente che provvederà alle dovute correzioni.

## 1.3 Architettura

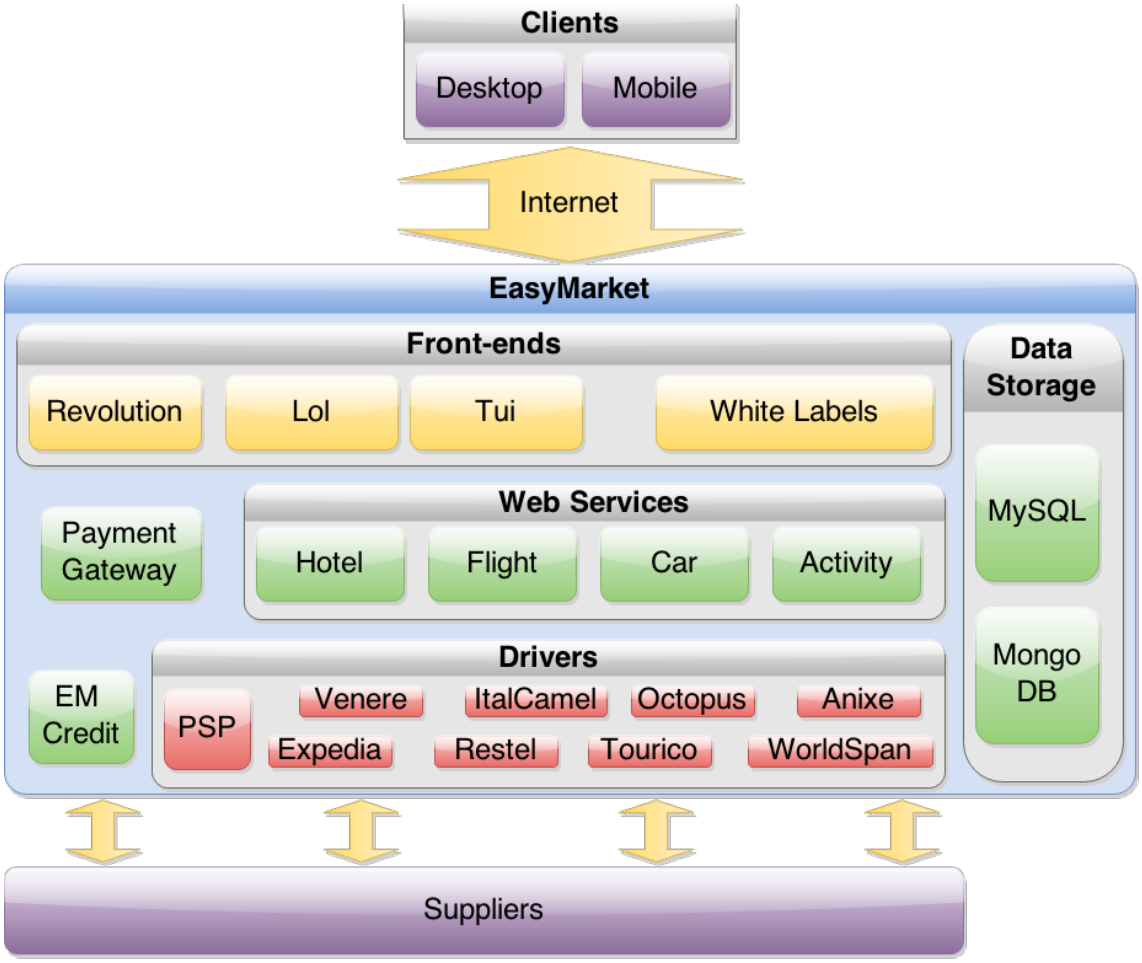


*Figura 1.4 : Schema architetturale di EasyMarket*

EasyMarket, attraverso la sua architettura informatica, vuole fornire un servizio di prenotazione online. Nelle prossime pagine sono descritti i principali componenti di quest'ultima e le problematiche che essa risolve.

**Multi-fornitura:** Come già detto, EasyMarket reperisce i propri prodotti

da diverse fonti. Queste fonti sono servizi di tipo REST, che sono un tipo di architettura software con un interfaccia che trasmette dati utilizzando il protocollo di rete HTTP. Il corpo dei messaggi ricevuti da questo tipo di servizi contiene dati formattati in XML o Json, anziché il classico HTML. Nel corso della tesi sarà descritto più nel dettaglio questo tipo di architetture. Questa soluzione è molto utilizzata quando si prevede che i client siano altre applicazioni, anziché esseri umani.

Ovviamente le modalità con cui i dati possono essere reperiti sono diverse per ogni servizio. Per comunicare con questi fornitori è necessario definire dei "driver". Si tratta di librerie create apposta per un certo prodotto. Il loro compito è consentire la comunicazione tra il fornitore e l'infrastruttura EasyMarket. Questo strato permette di ottenere offerte da più fonti contemporaneamente. Questi driver generalmente sono dei client HTTP progettati appositamente per "consumare" un determinato servizio REST.

**Multi-prodotto:** In figura 1.4 sono descritti una serie di componenti denominati "web-service". Implementano la logica di business dell'intera azienda. Ne esiste uno per ogni tipologia di prodotto. Il loro compito è rendere disponibile al resto dell'architettura, il prodotto per cui sono stati progettati. Ognuno di questi servizi utilizza i driver già descritti per interrogare i fornitori. I dati ricevuti, poi, devono essere "normalizzati". Normalizzare le offerte di un certo prodotto significa adattare le informazioni ad una struttura comune e omogenea. In più, in questa fase vengono svolte altre operazioni come il "pricing". Il prezzo di una data offerta va modificato secondo regole ben definite. Nel caso più semplice è necessario aggiungere il margine di guadagno di EasyMarket. Le agenzie che utilizzano Revolution hanno la facoltà di impostare i propri markup.

Anche questo influisce sul processo appena spiegato. Infine, attraverso una propria interfaccia REST, il web-service espone i dati processati. Questi servizi creano un livello di astrazione tra EasyMarket e i suoi fornitori. Omologano le informazioni ricevute e applicano tutte le trasformazioni del caso. Di fatto convertono il prodotto di terze parti in prodotto EasyMarket.

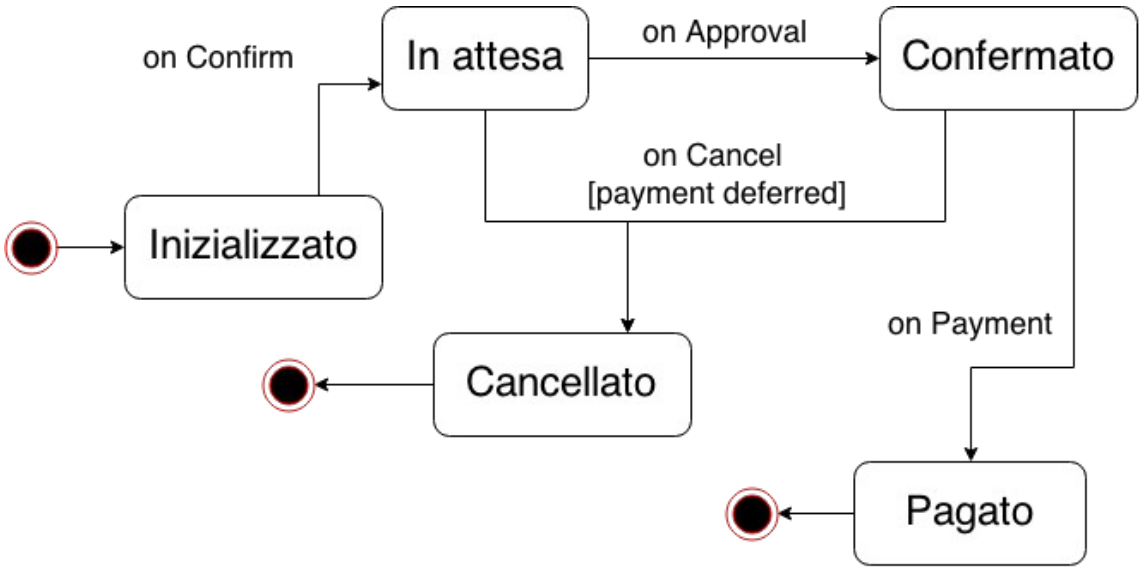


*Figura 1.5 : I diversi stati di una prenotazione*

**Gestione prenotazione:** un altro compito che deve assolvere un web-service è gestire tutto il processo di prenotazione. La figura 1.5 descrive gli stati attraverso cui un pratica transita. Il servizio deve gestire gli eventi che modificano questi stati e nello specifico fa da tramite tra il cliente e il fornitore. A fronte di un'operazione lato utente che cambia lo stato di una prenotazione, questa viene inserita nei log del sistema e notificata al fornitore interessato.

**Transazioni economiche:** allo stesso livello dei web-service, si trova il modulo di gestione delle transazioni economiche. Il “Payment Gateway” ha il compito di occuparsi dei flussi di denaro generati da EasyMarket. Viene richiamato dagli altri componenti dell'architettura ogni volta che devono

effettuare una operazione di tipo economico. Il suo funzionamento è molto simile a quello degli altri web-service. Il modulo vero e proprio raccoglie tutte le transazioni generate dagli altri componenti. I metodi di pagamento sono definiti per ogni utente. Con queste informazioni il payment gateway contatta i servizi bancari attraverso i cosiddetti "Payment Service Provider". Si tratta di servizi del tutto simili a quelli dei fornitori, infatti anche qui vengono definiti dei driver appositi. Questi servizi forniscono la possibilità di accedere ai circuiti di pagamento, come quelli delle carte di credito, ed effettuare i pagamenti. EasyMarket possiede anche un metodo di pagamento proprietario. Si tratta di una linea di credito chiamata "EM Credit". I clienti a cui è concessa hanno un fondo mensile a cui possono attingere per acquistare dall'azienda. Ogni mese i fruitori del servizio sono tenuti a saldare, anche con un semplice bonifico, il credito consumato.

**Presentazione prodotto:** L'interfaccia verso i clienti di EasyMarket è costituita dai suoi front-end. Sono applicazioni web che permettono all'utente di visualizzare le offerte, prenotare e pagare le proprie vacanze. Sono l'interfaccia grafica dell'intero sistema di prenotazione. Si occupano dell'interazione con l'utente e della "user-experience". Espongono le offerte richiamando i servizi sottostanti e permettono la gestione lato utente del ciclo di prenotazione. Sono la piattaforma dove vengono attuate strategie di marketing e campagne pubblicitarie. L'azienda possiede anche un servizio di prodotti "white label". Il loro nome significa letteralmente "senza etichetta" e sono un prodotto neutro, cioè non possiedono alcuna personalizzazione o stile. L'acquirente di tale prodotto deciderà in che modo "brandizzarlo"[2]. Nel caso di EasyMarket, il suo white label è un modello di front-end come quello in figura 1.6. Al momento dell'acquisto,

2 È l'atto di porre un marchio commerciale, o personalizzare, un prodotto o servizio in modo da renderlo riconoscibile.

insieme al cliente vengono definiti lo stile (colori, font ecc) e i loghi da inserire. Naturalmente anche in questo caso il prodotto esposto deriva dall'architettura sottostante.

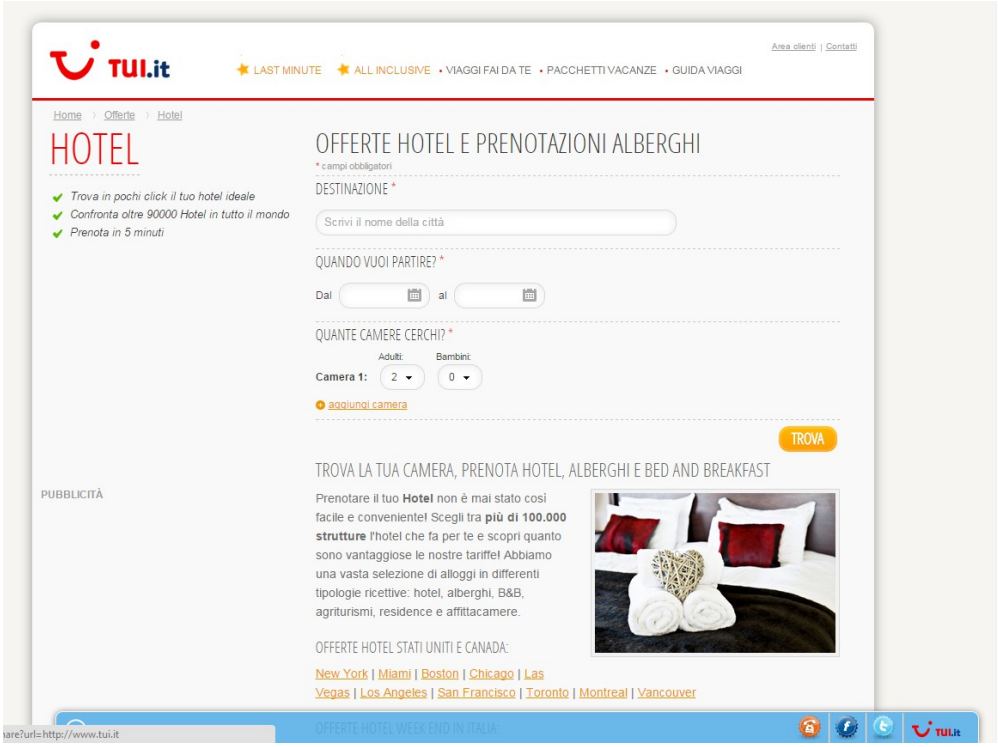


*Figura 1.6 : il sito tui.it è basato sul servizio di white label.*

**Caching e Persistenza:** Come tutte le applicazioni attuali, anche questa architettura ha bisogno di un sistema di salvataggio dati. Il DBMS più utilizzato in EasyMarket è MySQL [MyS15]. È un database relazionale open-source e multi piattaforma gestito da Oracle. La maggior parte dei moduli già visti si appoggia ad esso per persistere le proprie informazioni. Per alcuni particolari casi d'uso, tra cui il componente di cui parla questa tesi, è stato usato il db non relazionale MongoDB [Mon15]. È un database NoSQL basato sui documenti la cui forza è la velocità delle interrogazioni e la flessibilità dello schema dei dati.

Molti dei componenti di questa architettura implementano, tra la logica di business e lo strato di persistenza, dei sistemi di caching. Servono

soprattutto per evitare di sovraccaricare i database, ma anche per velocizzare il reperimento di certe informazioni. I dati salvati nelle cache sono file di configurazione e dati poco soggetti a cambiamenti durante il ciclo di vita delle applicazioni. Nonostante la loro utilità a volte è necessario resettare queste memorie temporanee. A questo scopo sono solitamente implementati anche metodi che cancellano il contenuto di esse.

### 1.3.1 Obiettivi qualitativi

L'architettura è stata così progettata per rispondere a certi requisiti di qualità.

**Scalabilità:** l'integrazione di un nuovo componente richiede solo che esponga una interfaccia REST ( o che la "consumi") e risulta trasparente per le parti non coinvolte nell'integrazione. Per questo motivo, ampliare il sistema risulta particolarmente semplice.

**Portabilità:** allo stato attuale molte tecnologie, librerie, e ambienti diversi sono utilizzati nel sistema e, a patto di mantenere le interfacce di comunicazione, ogni modulo può cambiare implementazione , base di dati sottostante e persino linguaggio, senza compromettere l'operatività generale.

**Manutenibilità:** dato che ogni componente costituisce un progetto a se stante, la rilevazione delle anomalie è più semplice e non produce errori in cascata. La manutenibilità è coadiuvata anche dall'utilizzo di tecnologie di versioning, di tracciamento dei bug e di gestione delle dipendenze.

**Evolvibilità:** il sistema è in continua evoluzione, ogni giorno vengono aggiunti fornitori ai web-service, aggiunte funzionalità ai front-end e inseriti nuovi dati nei database, il tutto senza bloccare l'operatività ne dover stravolgere l'architettura generale.

## 1.4 Produzione

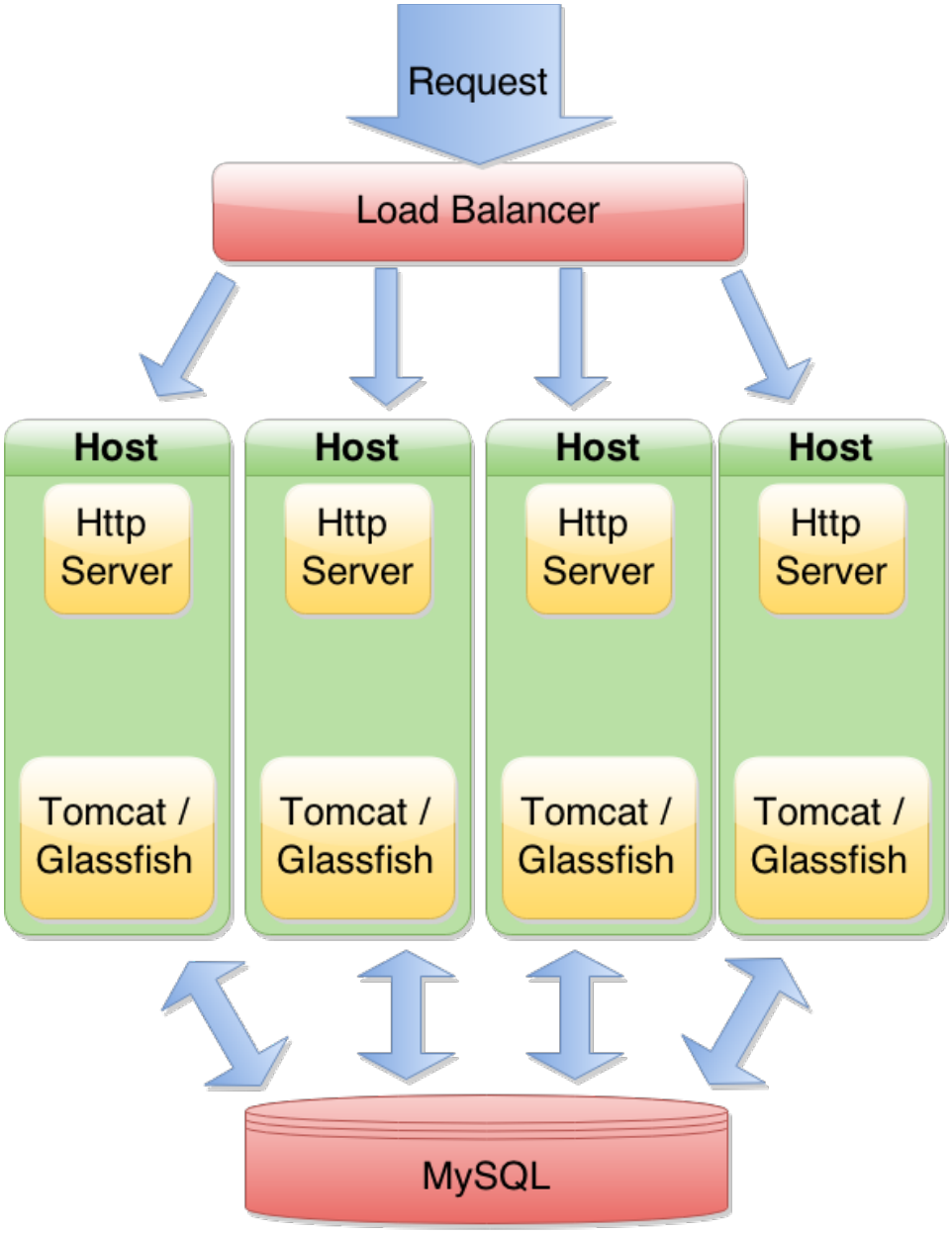


*Figura 1.7 : Architettura di produzione dei motori "hotel" e "voli"*

EasyMarket si serve di un cluster di server per rilasciare al pubblico i propri servizi. Su di esso è installato un hypervisor di tipo uno. Chiamato anche “virtual machine monitor”, è un software che crea e gestisce macchine virtuali. Gli hypervisor di tipo uno, detti anche nativi, sono installati direttamente sull'hardware e si occupano di gestire le risorse sottostanti. I sistemi operativi tradizionali sono eseguiti come processi su

questo sistema. Gli hypervisor di tipo 2 invece sono eseguiti su un sistema operativo tradizionale. Un sistema operativo installato su questi tipi di hypervisor viene astratto da quello sottostante [Hyp15].

Esistono molti vantaggi nella virtualizzazione che ha messo in atto EasyMarket. Le risorse sono ottimizzate, perché il software concede agli host solo le risorse necessarie. Su macchine fisiche questa gestione dinamica non è ovviamente possibile. Inoltre, grazie a questa amministrazione intelligente delle risorse, i costi di infrastruttura possono essere ridotti. È possibile creare infiniti ambienti virtuali, permettendo di scalare orizzontalmente l'intera architettura. Infine, grazie a strumenti di amministrazione appositi, si crea un sistema di controllo centralizzato ed efficiente.

IL software installato sul cluster dell'azienda è “VMware vSphere ESXi” [Vmw15]. Oltre alle funzionalità già descritte e ad un sistema di controllo integrato, vSphere fornisce funzionalità aggiuntive. Permette la migrazione “a caldo” delle macchine virtuali per evitare il “downtime” dovuto alla manutenzione dei server. Aumenta la robustezza dell'architettura gestendo le rotture degli host e i backup dei dati. In più consente di clonare completamente una macchina con tutte le sua applicazioni installate e i suoi dati.

Tutti le macchine virtuali del cluster di produzione hanno installato Debian. È una distro Linux nota per la sua leggerezza e stabilità, che la rende particolarmente adatta all'utilizzo in ambito server.

I prodotti più importanti venduti da EasyMarket sono hotel e voli. In figura 1.7 si vede come alla base dei ognuno dei servizi ci sia un istanza di MySQL. Al database si connettono quattro host gemelli su cui è eseguito il

web-service. Questa configurazione ridondante diminuisce la probabilità di interruzione del servizio. Anche se uno o più host si spengono, quelli rimanenti continuano a funzionare regolarmente. Gli host dedicati agli hotel hanno installato Tomcat come application server. [Tom15] definisce Tomcat come un web-server e servlet-container open-source sviluppato da Apache Software Foundation. Esso implementa diverse specifiche di livello enterprise di Java. Le più importanti sono Java Servlet, JavaServer Pages (JSP), Unified Expression Language (Java EL) e WebSocket. Come molti progetti open-source, Tomcat è mantenuto e aggiornato da una comunità di programmatori sotto la supervisione di Apache. Sugli stessi host è installato il server HTTP di Apache. Il suo compito è ricevere le richieste esterne, reperire le risorse chieste e formulare una risposta da inviare al client. Al di sopra delle macchine ridondate è inserito un bilanciatore di carico. Esso raccoglie le richieste provenienti da un unico dominio e le indirizza alle copie del web-service. La scelta della macchina a cui inoltrare la chiamata è dettata dal carico di quest'ultima e delle altre istanze. In questo modo il servizio non solo è ridondato ma è anche scalato in orizzontale, aumentandone la potenza complessiva. Per quanto riguarda il prodotto voli l'application server utilizzato è Glassfish. È l'implementazione di riferimento delle specifiche Java EE. Oltre a JS e JSP, implementa Enterprise Java Bean (EJB), Java Persistence API (JPA) e Java Message Service (JMS) [Gla15].

Il front-end Revolution e il servizio di pagamenti hanno strutture molto simili ai gestori di prodotto, ma con qualche differenza. Il carico di richieste a Revolution è minore rispetto ai web-service, indi per cui i server HTTP sono in numero minore rispetto agli application server. Il payment gateway invece gestisce la sua base di dati con un' istanza di SQL Server, il

DBMS proprietario di Microsoft.

## 1.5 Moduli principali

Questo paragrafo vuole descrivere più dettagliatamente alcuni dei moduli presentati nel paragrafo precedente. In particolare parlerà del motore hotel e di Revolution. La scelta è motivata dal fatto che il nuovo componente oggetto di questa tesi, è stato integrato proprio con questi due moduli.

### 1.5.1 Hotel web-service

Il motore di prenotazione hotel, come intuibile, si occupa di gestire le offerte delle strutture alberghiere reperite dai fornitori.

L'interfaccia REST che espone è basata sulla libreria Jersey. Essa implementa le API Jax-RS[3], che sono caratterizzate dal forte utilizzo di Java annotation. Queste notazioni vengono usate per mappare le richieste HTTP ai metodi opportuni. La figura 1.8 mostra un semplice metodo arricchito dalle annotation. La mappatura serve a determinare il metodo da richiamare a seconda della richiesta HTTP.

```
@Path("/users/{username}")
public class UserResource {

    @GET
    @Produces("text/xml")
    public String getUser(@PathParam("username") String userName) {
        ...
    }
}
```

*Figura 1.8 : Esempio di mappatura con Jersey.*

Il servizio comunica con parecchi fornitori, perciò definisce un buon numero di driver. Vengono utilizzati in parallelo, inoltrando contemporaneamente una richiesta a tutti i servizi esterni.

3 API Java per lo sviluppo di servizi REST

## 1.5.2 Revolution

E' un front-end dedicato al ramo B2B. Offre alle agenzie di viaggio un sistema completo di gestione delle prenotazioni previa sottoscrizione di un contratto con EasyMarket. Oltre al classico motore di ricerca, Revolution fornisce strumenti aggiuntivi rispetto ad un normale sito web dedicato ai privati.

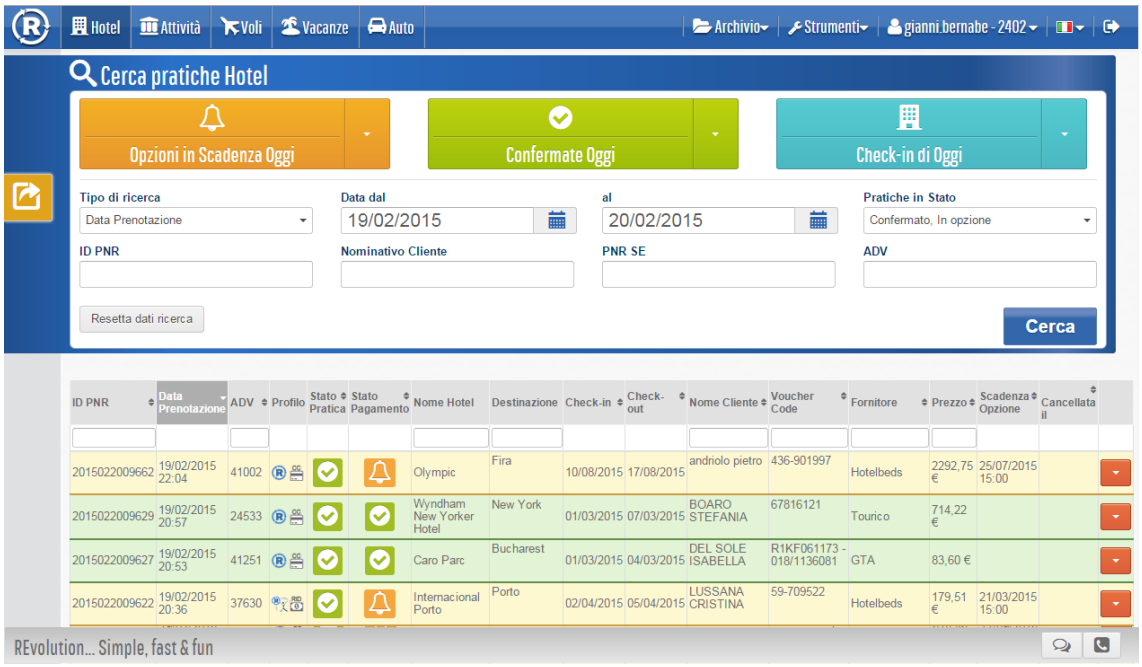


*Figura 1.9 : il back-office hotel di Revolution.*

Contiene un back-office, per la gestione del post-vendita (figura 1.9). Con questo strumento un'agenzia può tenere traccia delle proprie pratiche e, al bisogno, modificarle. Aiuta anche nella comunicazione con il cliente, fornendo la possibilità di scaricare i documenti e inviarli all'acquirente direttamente dall'applicazione. Nel caso di gruppi di agenzie (network), sono presenti gli strumenti per gestire anche queste realtà. È possibile gestire le affiliate al proprio network con un sistema di autorizzazioni. Gli amministratori di tali gruppi possono decidere quali permessi concedere e quali revocare, fino al livello di singolo dipendente.

Revolution è l'unica interfaccia da cui è possibile consultare tutti i prodotti di EasyMarket. Altri front-end si limitano a solo ad una parte di esso (hotel, voli, ecc.).

Tutto il progetto è basato sul framework per Java, Spring, che sarà illustrato più approfonditamente nel corso della tesi. È un framework di livello enterprise[4] che fornisce molte funzionalità comuni agli applicativi sviluppati al giorno d'oggi.

L'interfaccia di Revolution segue il modello architetturale MVC. Questo modello prevede il disaccoppiamento tra presentazione (View) e contenuti (Model). Il compito di unire queste due entità per creare l'interfaccia grafica spetta ai componenti dell'applicazione specializzati in questo compito (Controller). Revolution implementa la presentazione attraverso dei modelli di pagine web scritti con un linguaggio di template. Tra quelli supportati da Spring, è stato scelto Velocity [Vel5]. Si tratta di un motore di template sviluppato da Apache Software Foundation. Fornisce un linguaggio potente per referenziare oggetti Java. Il suo scopo è separare ulteriormente la presentazione dal livello di business.

## 1.6 Tecnologie utilizzate

Questo paragrafo approfondisce le tecnologie già citate usate in EasyMarket. Oltre a ciò saranno anche introdotti gli strumenti di supporto allo sviluppo adottati.

### 1.6.1 Linguaggio

Tutti i componenti dell'architettura EasyMarket sono scritti in Java. Alcuni di loro, come Revolution, usano Spring per definire la propria

4 Un software di livello enterprise presenta funzionalità e qualità di livello superiore, adatte all'utilizzo industriale.

struttura interna. Spring è composto di tanti moduli, ognuno progettato per affrontare un determinato problema.

**Core:** questo modulo contiene le funzionalità chiave che definiscono il framework stesso. La più importante è sicuramente la dipendency-injection. Normalmente ogni entità, nella programmazione ad oggetti, definisce le proprie dipendenze al più tardi, a livello di compilazione. La dipendency-injection solleva le classi dalla definizione di tali dipendenze e delega questa operazione a run-time. Ogni classe non conoscerà la reale implementazione delle proprie dipendenze fino all'esecuzione del programma. Questa opportunità permette di creare architetture "loosely coupled", i cui oggetti possono essere rimossi, aggiunti o cambiati senza generare errori in quelli che li utilizzano.

**Web:** come Jersey, anche Spring ha la propria implementazione delle architetture REST e MVC. Fornisce inoltre un client HTTP di semplice utilizzo.

**Data:** si tratta di un gruppo di moduli volti a gestire l'interazione tra database e applicazione. Ogni modulo si occupa di una specifica tecnologia o database. Quelli usati in EasyMarket sono:

- MongoDB: come intuibile, fornisce supporto per l'utilizzo di tale database. Espone una serie di funzionalità, costruite sulla base del driver nativo per Java fornito da MongoDB. Consente di trasformare qualsiasi oggetto in documento grazie a delle Java annotation specifiche. In più fornisce un interfaccia con i metodi più comuni di accesso ai dati.
- Java Persistence API: è una specifica standard di Java che

implementa il concetto di ORM[5]. Quest'ultimo è il processo di trasformazione di un istanza di una classe in un record di un database relazionale. Spring ha sviluppato la propria versione di queste specifiche, fornendo tutti gli strumenti per gestire al meglio le interrogazioni e lo schema dei dati.

**Security:** è il modulo standard per mettere in sicurezza applicazioni Spring-based. Per applicazioni "standalone" supporta l'autenticazione e l'autorizzazione. Nei progetti web-based sono previsti anche meccanismi di controllo degli accessi basati su URL.

### 1.6.2 Basi di dati

Come già detto, il database più utilizzato in EasyMarket è MySQL. Tra le catteristiche che [MyS15] elenca le più importanti sono:

- Multi-piattaforma: installabile su tantissimi sistemi operativi diversi.
- Multi-thread: fa uso di thread a livello di kernel, sfruttando al meglio le risorse di calcolo.
- Join veloci: grazie ad un algoritmo nested-loop ottimizzato.
- Fornito come programma separato per essere usato in ambienti dotati di connettività. In alternativa è disponibile come libreria integrabile in altre applicazioni.
- Sicurezza: possiede un sistema di autenticazione e di privilegi sicuro e flessibile. In più supporta la crittografia.
- Scalabilità: adatto alla gestione di enormi banche dati. Alcuni utenti lo usano con 200.000 tabelle e circa 5 miliardi di record.
- Strumenti: include diversi programmi di utilità e client come ad

5 Object Relational Mapping.

esempio MySQL WorkBench [MyW15] e phpMyAdmin [Php15].

In alcuni moduli i dati sono memorizzati utilizzando il database non relazionale MongoDB. Questo db è il tema centrale di questa tesi per cui si rimanda al paragrafo al lui deidicato per un' argomentazione completa.

### 1.6.3 Supporto allo sviluppo

Il team di sviluppo di EasyMarket utilizza delle tecnologie e delle applicazioni che accellerano la produttività e aiutano durrante lo sviluppo. Questi strumenti di supporto possono essere suddivisi in tre categorie.

**Gestione delle dipendenze:** gli applicativi aziendali fanno largo uso di librerie esterne. Manualmente sarebbe impossibile gestirle e aggiornale. Per questo si fa uso di un gestore delle dipendenze. Per Java esiste Apache Maven [Mav15]. Basato sul cocetto di "Project Object Model" (POM), Maven gestisce in automatico le versioni delle librerie  grazie ad una sistema di repository locali e in rete. Un progetto Maven contiene sempre

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.1.5.RELEASE</version>
    </dependency>
</dependencies>
```

*Figura 1.10 : esempio di dipendenza dichiarata nel file pom.xml.*

un file XML (pom.xml) in cui definire le proprie dipendenze (figura 1.10). In più, con opportuni plugin è possibile aggiungere funzionalità come la generazione di documentazione e report.

**Versioning:** I programmatori, quando lavorano in team possono incorrere in vari problemi. Ognuno ha la propria copia dei codici su cui

lavorare. Questo implica che ogni modifica apportata non sarà presente sulle copie dei propri colleghi. Ancora più grave è la situazione in cui due sviluppatori devono modificare la stessa porzione di programma. È molto difficile tenere sotto controllo questi problemi senza strumenti ausiliari.

Il controllo dei sorgenti permette di risolvere questi e altri problemi di un team di sviluppo. Anche detto version control, si tratta della gestione delle modifiche a documenti, programmi, grossi siti web e altre collezioni di informazioni. Queste modifiche sono di solito identificate da un codice alfanumerico detto "versione". Il concetto di versione è generalmente visto come una linea di sviluppo (trunk) con biforcazioni (branch) che possono eventualmente ricongiungersi con il filone principale [Ver15]. Ciò crea un grafo acicliclo orientato, come in figura 1.11.

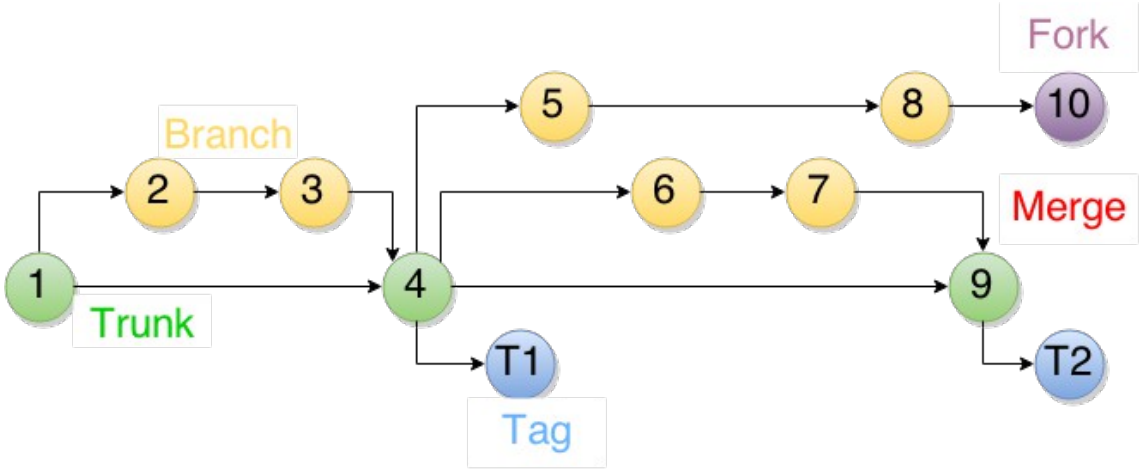


*Figura 1.11 : fonte [Ver15].*

Il controllo delle revisioni è affidato ad un applicazione che consente diverse azioni come:

- Revert: tornare ad una versione più datata di un documento.
- Update: richiedere una versione più recente.
- Commit: rendere pubblica una nuova modifica.

- Merge: fondere due branch di uno stesso documento.
- Conflict resolution: gestire le eventuali versioni incompatibili fra loro.

EasyMarket utilizza il source control grazie ad un progetto di Apache Software Foundation: Apache Subversion [Sub15].

**Bug tracking:** Il tracciamento dei bug è affidato ad una applicazione di terze parti. Un bug traking system è un applicativo che tiene traccia degli errori all'interno dei software. Solitamente ne fornisce una descrizione e alcuni dettagli per renderli più facilmente risolvibili [Bug15].

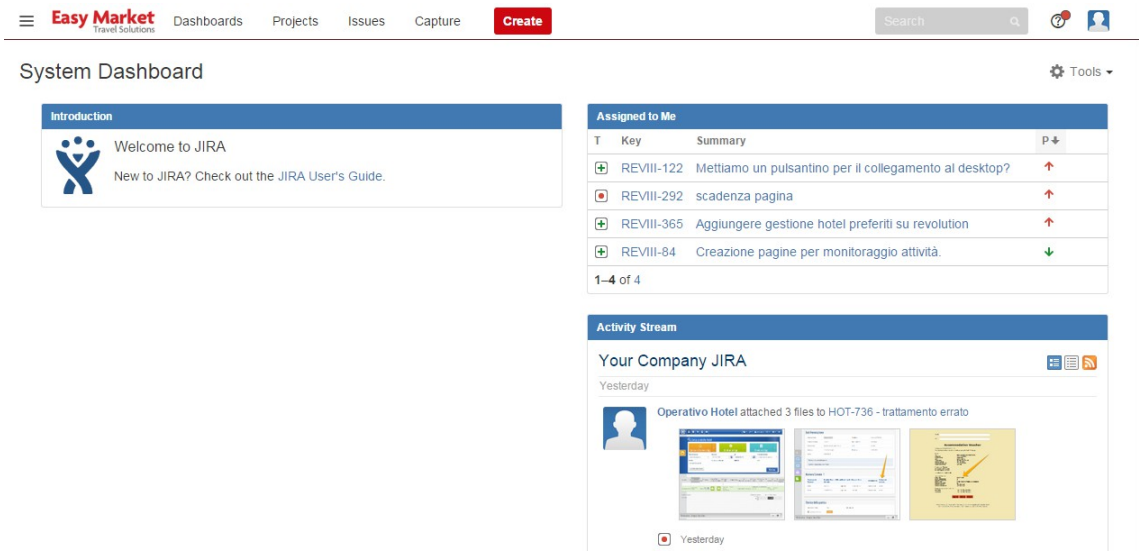


*Figura 1.12 : Pagina principale di Atlassian Jira.*

In figura 1.12 è mostrata la home page del servizio di bug tracking Atlassian Jira [Jir15]. È una piattaforma personalizzabile a seconda dell'ambiente d'uso. Il sistema si occupa di organizzare i bug secondo parametri come la gravità dell'errore, il progetto di appartenenza, la persona responsabile della correzione ecc. Permette di seguire il ciclo di vita dei bug, dalla prima notifica, alla correzione (o chiusura). In aggiunta permette a tutte le figure interessate di comunicare tra loro per raggiungere la

miglior soluzione nel minor tempo possibile. EasyMarket si è affidata ad Atlassian  per gestire gli errori dei suoi applicativi.

# Capitolo 2

# NoSQL

In questo capitolo saranno discusse le principali caratteristiche del movimento NoSQL, partendo dalla sua storia. Saranno messi in luce i limiti del modello relazionale e ciò che implicano. Verrà esposto il teorema CAP e le due principali filosofie di persistenza dei dati, ACID e BASE saranno messe a confronto. Infine verranno introdotte le categorie di database appartenenti al movimento NoSQL.

## 2.1 Storia

Il termine NoSQL venne usato la prima volta nel 1998 da Carlo Strozzi per indicare un database relazionale che non sfruttasse il linguaggio di interrogazione SQL [Str98]. Tale termine fu successivamente reintrodotto nel 2009 da un impiegato di RackSpace, Eric Evans, durante un convegno organizzato da Johan Oskarsson di Last.fm per discutere dell'utilizzo di basi di dati distribuite open-source.

NoSQL è l'acronimo di "Not Only SQL", lasciando intendere che il movimento non è inteso come un rifiuto categorico del modello relazionale ma vuole solo essere una valida alternativa in un settore monopolizzato fino a quel momento.

I database relazionali, per trent'anni, sono stati i leader indiscussi del mondo del "data storage", tuttavia nel tempo si sono presentati problemi che hanno spinto gli sviluppatori verso soluzioni diverse. Lo slancio in

favore dell'approccio NoSQL è venuto soprattutto dalle grandi aziende.

Google ha pubblicato nel 2006 un documento intitolato "BigTable : A Distributed Storage System for Structured Data" [CDG06]. Dal documento stesso si può leggere:

"*Big Table è un sistema di memorizzazione distribuito per gestire dati strutturati che sono progettati per scalare a dimensioni molto grandi: Petabyte di dati attraverso centinaia di server.*"

Il volume dei dati che Google doveva immagazzinare era talmente grande che i normali database in commercio non riuscivano nell'impresa, se non a costi esorbitanti. Per questo l'azienda ha progettato un sistema proprietario che scalasse facilmente in orizzontale e desse ai client il controllo sul formato e la struttura dei dati.

Nel 2012 Amazon rilasciò la prima versione di DynamoDB [Dyn15]. Anche il colosso di Seattle si è trovato a non riuscire più a gestire la mole dei propri dati con i DBMS classici, così costruì il proprio database. Lo progettò in modo che fosse flessibile scalabile e distribuito.

## 2.2 Limiti del modello relazionale

Come le grandi compagnie avevano già capito, i database relazionali non erano onnipotenti. Negli anni, lo sviluppo esponenziale del web ha messo a dura prova l'approccio classico alla persistenza. Esso presenta diversi limiti.

**Join:** sono classiche operazioni dei db tradizionali. Permettono di mettere in relazione i dati di tabelle diverse nello stesso database ma hanno un costo computazionale piuttosto elevato. Gli esperti hanno speso molte energie cercando gli algoritmi migliori, i più veloci. Spesso i DBMS offrono dei sistemi in grado di determinare il miglior piano di esecuzione di

un'interrogazione. Tutto questo per ottimizzare e minimizzare il tempo di esecuzione. Nonostante tutti questi sforzi, il join rimane una operazione altamente dispendiosa, che non scala bene all'aumentare dei record e degli accessi al db.

**Conflitto di impedenza:** è un insieme di difficoltà tecniche che si incontrano quando si cerca di adattare la struttura tabellare dei database alle strutture dati dei linguaggi di programmazione. Questo si verifica specialmente con i linguaggi object-oriented. Come già accennato nel primo capitolo, spesso si fa uso una tecnica chiamata ORM per mappare le tabelle agli oggetti applicativi. Questa però richiede uno sforzo aggiuntivo agli sviluppatori per creare la mappatura adatta.

**Scalabilità:** i database relazionali hanno sempre avuto difficoltà a scalare orizzontalmente, in quanto sono stati originariamente realizzati per applicazioni centralizzate. È comunque possibile scalare in verticale, ma è più complesso e costoso. Oltrettutto questa soluzione ha anche dei limiti tecnologici, che la soluzione orizzontale non presenta.

**Schema rigido:** le tabelle di uno schema realzionale sono definite in fase di progettazione. Ciò rende molto difficile modificarle in un secondo momento, senza rischiare perdite di dati.

Le nuove applicazioni, come quelle basate sul web, proprio per colpa di questi limiti hanno bisogno di un approccio diverso. NoSQL è la risposta a questo problema. Questo nuovo approccio risponde ai requisiti che le nuove architetture cercano in un sistema di memorizzazione.

**Semplicità:** è uno degli elementi fondamentali che permette ai NRDBMS[6] di scalare in maniera così efficiente. Non essendo di base

6 Not Relational DBMS

troppo complessi, è possibile creare infrastrutture distribuite in modo relativamente semplice. Molti di essi permettono di aggiungere nodi a caldo in modo assolutamente trasparente all'utente finale. La sintassi delle query è più semplice grazie all'assenza di tabelle e join.

**Orientato agli oggetti:** lo sviluppo di applicazioni basate su NoSQL non ha bisogno di definire una mappatura tra il database e le applicazioni stesse. Le entità di un db non relazionale possono essere strutturate come una rappresentazione diretta degli oggetti del dominio.

**Minor costo:** l'utilizzo di un db NoSQL è più economico sotto diversi punti di vista. Richiede meno risorse hardware per compiere il proprio lavoro, quindi meno costi di infrastruttura. Il mantenimento dei dati al suo interno è meno macchinoso dei database relazionali. Le complesse operazioni che servono solitamente per aggiornare la struttura di una tabella, in questi contesti non servono. In virtù di ciò anche il costo in termini di personale e tempo di sviluppo risulta ridotto.

**Replicazione e distribuzione:** la replicazione permette di creare copie dei dati e mantenerle su diverse istanze di un database. Le istanze comunicheranno tra loro per mantenere aggiornata ogni singola copia assicurando una maggiore tolleranza ai guasti. Un database distribuito invece suddivide i propri dati su diverse macchine, aumentando le risorse di calcolo e di memoria disponibili.

**Flessibilità:** la mancanza di uno schema predefinito rende i db NoSQL estremamente flessibili. Possono salvare dati molto eterogenei e variabili senza alcun tipo di problema.

## 2.3 Il teorema CAP

Il teorema CAP è stato formulato per la prima volta da Eric Brewer nel 2000 e ancora oggi ne porta il nome[7]. Tale teorema fu dimostrato nel 2002 da Gilbert e Lynch del MIT[8][GL02] ed afferma che un sistema distribuito può al massimo soddisfare contemporaneamente due delle tre seguenti caratteristiche:

- **Consistency:** tutti i nodi posseggono gli stessi dati, e dopo ogni modifica, essi riflettono tale modifica.
- **Availability:** il sistema risponde sempre a tutte richieste ricevute. In altre parole esso è sempre disponibile.
- **Partition Tolerance:** è la capacità di un sistema di resistere e recuperare perdite di dati o guasti. Un guasto in un nodo non dovrebbe far crollare l'intero sistema.

7 Il teorema CAP è detto anche teorema di Brewer
8 Massachussets Institute of Tecnology

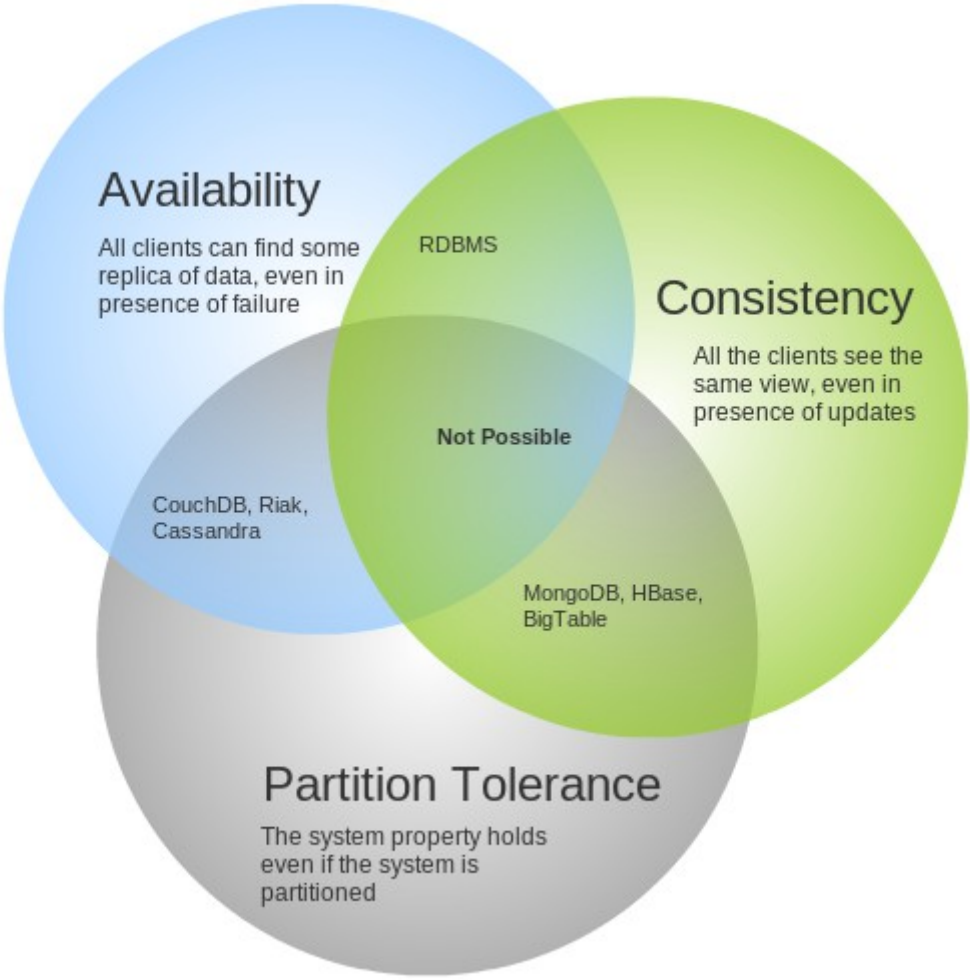


*Figura 2.1 : Diagramma di Venn rappresentante il teorema CAP*

La figura 2.1 è una rappresentazione grafica di quanto appena detto.

I vincoli posti da questo teorema sono stati il caposaldo per l'ascesa dei recenti sistemi non relazionali. I nuovi sistemi su larga scala e distribuiti hanno spesso bisogno di availability e partition tolerance, perciò la consistenza deve essere necessariamente messa in secondo piano. I database relazionali, che supportano consistenza e disponibilità dei dati non sono più la soluzione migliore, o comunque non l'unica.

## 2.3 ACID e BASE

Come già spiegato, i database relazionali fanno dell'integrità e disponibilità dei dati la propria forza. Per offrire queste caratteristiche si basano su una serie di proprietà delle loro transazioni note come ACID. É un acronimo che significa:

- **Atomicità:** un transazione è un operazione atomica, o è eseguita completamente o non è eseguita affatto.
- **Consistenza:** il database si trova in uno stato consistente prima e dopo l'esecuzione di una transazione.
- **Isolamento:** ogni transazione è indipendente dalle altre, l'esecuzione di una non deve alterare l'esecuzione di un altra.
- **Durabilità:** detta anche persistenza. I cambiamenti apportati al database non devono andare persi. Di norma per impedire la perdita di dati i db mantengono dei file di log da utilizzare in caso di guasti.

Come si può vedere, ACID è un approccio molto rigoroso, che pone molta attenzione alla availability e alla consistency.

Il teorema CAP, come già discusso precedentemente, impedisce ai database basati su ACID di assicurare partition tolerance. È per questo motivo che sono così inadatti ad ambienti distribuiti. Al giorno d'oggi però, i sistemi distribuiti sono una realtà di tutti i giorni e NoSQL li supporta grazie ad un approccio diverso da ACID.

BASE è l'approccio usato da tutti i sistemi non relazionali esistenti al momento. Esso si basa su tre principi:

- **Basic Availability:** la disponibilità dei dati è garantita. Invece di mantenere un singolo grande bacino di dati, concentrandosi sulla

resistenza ai guasti dello stesso, si preferisce spargere i dati su più sistemi di salvataggio con un alto grado di replicazione. In questo modo si crea un infrastruttura intrinsecamente “fault tolerant”, senza bisogno di complessi sistemi. Questa distibuzione però non assicura la consistenza dei dati.

- **Soft State:** lo stato del sistema può cambiare nel tempo. Non c'è alcuna garanzia che i dati rimangano invariati, anche in assenza di modifiche. Questo principio, in pratica, lascia allo sviluppatore, l'onere di assicurare o meno la consistenza dei dati.
- **Eventual Consistency:** il sistema garantisce che, in un certo punto nel futuro, i dati convergeranno ad uno stato consistente, senza garanzie su quando questo accadrà. Con lo stesso principio i dati saranno propagati in tutto il sistema. Non sono effettuati controlli sulla consistenza tra una transazione e l'altra.

Si nota come, al contrario dei RDBMS, i sistemi BASE si focalizzino sulla disponibilità del servizio anzichè sulla consistenza.

Lo scopo dei database non relazionali è infatti fornire una base di dati sempre disponibile ed espandibile a piacere, anche se ciò significa non garantire dati sempre aggiornati.

## 2.4 Tassonomia dei database NoSQL

In questo paragrafo sono elencate le categorie in cui si suddividono i database NoSQL. Anche se tutti rispettano i dettami logici, il mondo non relazionale presenta una gran varietà di db, racchiusi in quattro categorie.

**Key-value:** sono i database con il modello dei dati più semplice e intuitivo. Salva semplicemente delle coppie chave-valore. Le chiavi sono

mappate all'interno di contenitori logici chiamati "bucket". I bucket non raggruppano fisicamente i dati, servono solo in fase di lettura. Viene usata una tabella hash per mappare le chiavi all'interno di ogni bucket. Per leggere un valore è necessario conoscere la chiave e il bucket in cui si trova. Non avendo una sintassi di interrogazione a parte quella già descritta, questo tipo di database non sono l'ideale se si necessita di interrogare o aggiornare i dati. Secondariamente, all'aumentare del volume dei dati, mantenere un insieme di chiavi uniche può diventare complicato. Il già citato DynamoDB è parte di questa categoria.

**Document-based:** si tratta ancora di collezioni di coppie chiavi-valore. La differenza nei database basati sui documenti è che gruppi di chiavi possono essere raggruppati in strutture chiamate appunto, documenti. Grazie a questa sovrastruttura, è possibile definire un metodo per cercare i documenti che rispondono a certi criteri. MongoDB, che sarà approfondito nel corso della tesi, fa parte di questa categoria.

**Column-based:** questi database, in contrapposizione ai database relazionali "row-oriented", memorizzano i dati per colonne anzichè per righe. Queste colonne sono raggruppate in strutture chiamate "famiglie" che ricordano molto il concetto di tabella. La differenza sostanziale da una tabella relazionale è che una famiglia non definisce a priori le colonne al suo interno. Le righe in una famiglia possono avere un numero differente di colonne l'una dall'altra. L'organizzazione per colonne ha un alto impatto a livello di performance. I valori in una stessa colonna sono contigui su disco. In fase di ricerca tutti i valori di una colonna vengono caricati in memoria contemporaneamente, rendendo le interrogazioni più veloci. BigTable di Google, di cui si è già parlato è basato su colonne.

**Graph-based:** si basano sulla teoria dei grafi, quindi utilizzano i concetti

di:

- **Nodi:** rappresentano le informazioni principali e più importanti.
- **Archi:** rappresentano attraverso informazioni secondarie le relazioni tra i nodi.

I database basati su grafo possono essere visti come un caso speciale o un utilizzo di quello document-oriented. Basta delegare alcuni documenti alla rappresentazione delle relazioni tra gli altri. Un utilizzo quasi scontato per questi database sono i social network. Un aspetto interessante è la possibilità di sfruttare gli algoritmi dei grafi per ottimizzarne l'utilizzo.

## 2.5 Svantaggi del modello non relazionale

Sono già stati spiegati i punti di forza dei database non relazionali. Anche questi però non sono esenti da mancanze. NoSQL non è la soluzione ultima a tutti i problemi di persistenza dei dati. Un post di Nicholas Greene[Gre13] illustra alcuni dei limiti del modello non relazionale.

**Consistenza dei dati:** NoSQL assicura la possibilità di scalare agevolmente in orizzontale e di replicare i dati in diversi server. Facendo riferimento al teorema CAP illustrato in precedenza, questo assicura availability e partition tolerance. L'integrità dei dati, quindi deve necessariamente essere messa in secondo piano.

**Standardizzazione:** la natura open-source di questi sistemi può essere considerata il loro più grande punto di forza e di debolezza allo stesso tempo. Questo perchè esistono molte implementazioni, ognuna diversa dall'altra. Ancora non si può fare affidamento su uno standard come lo è SQL per il modello relazionale. L'integrazione in sistemi presistenti può richiedere grossi sforzi. In più una migrazione da un database NoSQL può

creare non poche difficoltà. È necessario fare le dovute considerazioni quando si pensa di adottare questi database.

La filosofia stessa del movimento NoSQL non pretende di soppiantare il modello relazionale. Vuole essere solo un' alternativa per quelle applicazioni a cui un database tradizionale non è adatto.

# Capitolo 3

# Tecnologie

Questo capitolo illustra le tecnologie e gli strumenti utilizzati per sviluppare il nuovo componente di cui parla questa tesi. Sarà approfondito il database non relazionale MongoDB già accennato nei capitoli precedenti. Oltre a questo saranno descritte le altre tecnologie che fanno parte del modulo. Tra queste la tecnologia Java per lo scambio di messaggi, JMS, e le principali architetture software usate in ambito web.

## 3.1 MongoDB

MongoDB è un database basato sui documenti. È l'esponente del mondo NoSQL utilizzato da EasyMarket nella propria architettura.

Secondo [Mon15] le funzionalità chiave fornite da questo DBMS sono:

- **Alte performance:** il modello dei dati con strutture annidate riduce le operazioni di I/O sul sistema. Gli indici migliorano ulteriormente le performance in lettura.
- **Alta affidabilità:** MongoDB assicura alta affidabilità attraverso i suoi replica set. Essi garantiscono ridondanza dei dati e failover automatico.
- **Automaticamente scalabile:** è possibile distribuire i dati attraverso un cluster di macchine grazie allo sharding. In

ambienti a bassa latenza, i replica set possono migliorare il throughput delle letture.

MongoDB codifica i propri dati in BSON (Binary Json), un formato più espressivo del classico Json. Allo stato attuale esistono client driver, per tutti i principali linguaggi di programmazione e sviluppati direttamente dal team di MongoDB. Rimanendo fedele alla propria natura "BASE", questo DBMS non supporta transazioni ACID. Ogni transazione coinvolge uno e un solo documento. Le transazioni sono isolate secondo il concetto di read uncommitted. Non esistono transazioni o isolamento per più documenti.

### 3.1.1 Modello dei dati

I dati su MongoDB, come ogni database non relazionale, hanno uno schema flessibile. Il modello ruota attorno al concetto di documento. Un documento è un entità composta di coppie chiave-valore, detti anche campi. Il loro valore può essere a sua volta un altro documento o una collezione (figura 3.1).


*Figura 3.1 : Esempio di documenti annidati.*

Ogni documento contiene almeno un campo, che lo identifica univocamente all'interno di una collezione[9]. Le collezioni sono insiemi di documenti che di solito hanno struttura o semantica simili, tuttavia queste possono contenere qualsiasi documento indistintamente, senza vincoli. I vantaggi di questa organizzazione sono evidenti:

- Gli oggetti di un dominio applicativo possono avere la propria rappresentazione fedele nel database.
- I documenti e gli array annidati riducono il bisogno di costosi join.
- Lo schema dinamico dei documenti supporta il polimorfismo.

MongoDB consente di referenziare un documento senza per forza annidarlo. Esistono due modi:

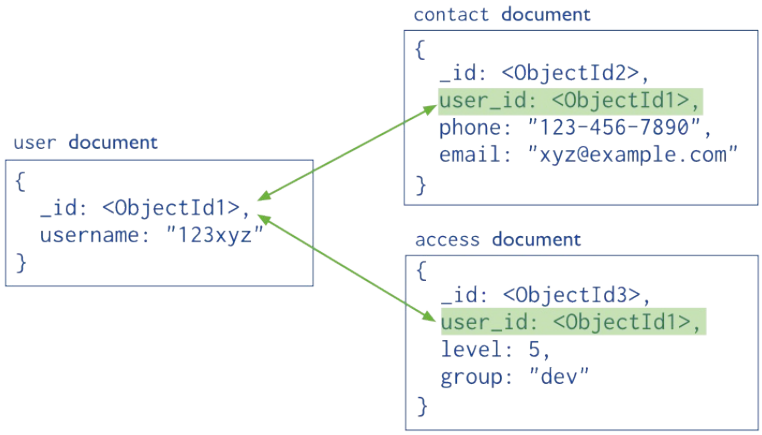


*Figura 3.2 : Esempio di dati normalizzati*

- Manual Reference (figura 3.2): aggiungendo ad un documento una chiave che contenga l'id di un altro documento è possibile replicare il concetto di chiave esterna. E' un approccio molto

9 Attenzione alla differenza tra “collezione” (insieme di documenti) e “collezione” (insieme di valori)

"naif" in quanto non prevede nessun controllo di integrità della referenza e i due documenti devono stare nella stessa collezione.

- DBRef: Il documento innestato è sostituito da un suo "segnaposto", un sottodocumento contentente id, collezione e database dell'entità originale. Anche in questo caso non esistono controlli sull'integrità dei dati. La risoluzione di tali dipendenze è a carico dell'applicazione. I driver forniti da MongoDB hanno metodi che aiutano in questo compito.

Le scritture su MongoDB sono atomiche a livello di documento, e nessuna operazione atomica interessa più di un documento. Una struttura di dati annidata favorisce molto le scritture. Al contrario entità normalizzate (che usano referenze) costringono ad eseguire multiple scritture, rinunciando all'atomicità. Nonostante ciò, entità normalizzate sono molto più flessibili di entità annidate. È quindi necessario fare le dovute considerazioni, e progettare uno schema bilanciato tra flessibilità e atomicità.

Un' altra considerazione da fare sull'uso di dati "normalizzati" e non è la "crescita del documento". Aggiungere campi ad un documento ne aumenta la dimensione. MongoDB attua una strategia di "pre-allocazione". Alloca automaticamente più spazio di quello necessario per minimizzare la probabilità di dover riallocare i dati. Un applicazione che modifica, aggiunge e rimuove spesso dati può portare MongoDB ad occupare molto spazio disco inutilmente. I dati "normalizzati" aiutano a tenere sotto controllo questo scenario.

Esistono comunque operazioni di deframmentazione per recuperare spazio inutilizzato.

### 3.1.2 Indicizzazione

Anche se la classica definizione di NoSQL non prevede l'uso di indici, MongoDB li supporta come un qualsiasi db relazionale. Sono definiti a livello di collezione. In un contesto di “big-data”, è impensabile scorrere un intero insieme di documenti per eseguire una ricerca. Oltretutto una parte di questi documenti potrebbe non c'entrare nulla con la query in esecuzione. La mancanza di indici obbligherebbe a considerare anche questi documenti “inutili”. In ambienti non relazionali come questo un buona progettazione degli indici può fare veramente la differenza. [Mon15] consiglia di creare indici che supportino le più comuni interrogazioni, così che venga considerato l'insieme di dati più piccolo possibile. MongoDB può trarre vantaggio dagli indici anche in altri modi:

- Risultati ordinati: ordinare una query per un campo su cui è definito un indice non richiede ordinamenti aggiuntivi. Da notare che ogni indice è attraversabile in direzione crescente e ascendente.
- Copertura dei risultati: se la proiezione di un'interrogazione include solo campi indicizzati, non sarà necessario esaminare nessun documento. I dati saranno presi direttamente dagli indici.

Tutti gli indici sono implementati come B-tree e possono essere di di diversi tipi

**Single Field:** MongoDB supporta indici su qualsiasi campo di un documento. Tutte le collezioni hanno almeno un indice sul campo “_id” indice.

**Compound:** gli indici composti si comportano come indici singoli. Ci sono però delle sostanziali differenze.


*Figura 3.3 : Esempio di indice composto*

Prendiamo ad esempio l'indice in figura 3.3; le interrogazioni che ne traggono vantaggio sono quelle che includono:

- il campo “item”
- il campo “item” e “location”
- il campo “item” e “stock”. In questo ultimo caso però, sarebbero più efficienti due indici distinti sui due campi.

MongoDB non trae vantaggi nel caso di query che includono:

- solo il campo “location”
- solo il campo “stock”
- una combinazione dei due

In definitiva, gli indici composti di MongoDB sono utili quando sono definiti su campi spesso interrogati insieme oppure nel caso in cui un indice semplice non basti. Si noti che l'ordine dei campi nella dichiarazione dell'indice ne definisce l'importanza. I campi che si considera più importanti quindi andranno dichiarati prima.

**Multikey:** per indicizzare un campo di tipo array, viene creata una lista ordinata contenente tutti gli elementi del vettore. MongoDB decide autonomamente se creare un indice multikey, se vede che il campo è una collezione, senza bisogno di esplicitarlo. Putroppo questa tipologia ha delle

limitazioni. Non è possibile, per esempio indicizzare vettori paralleli. In sostanza un indice composto può contenere al più un campo di tipo array. Provando ad inserire un documento che violi questa regola si otterebbe un errore e la scrittura fallirebbe. Esistono altre limitazioni (vedi indici hashed e paragrafo 3.1.4).

**Text:** è un tipo di indice che supporta la ricerca per campi di tipo stringa o vettore di esse, fatto appositamente per interrogazioni che usino l'operatore $text. Quest'ultimo consente la ricerca di termini o frasi all'interno di documenti. Anche questo tipo di indici soffre di restrizioni. Non aiuta in alcun modo l'ordinamento, neanche se parte di un indice compound. Se un indice composto ne contiene uno "text", non è possibile utilizzare altri indici speciali (es. multikey).

**Hashed:** come da nome, questo indice elabora ogni documento secondo una funzione hash. Non supporta query di intervallo, ma è comunque possibile definire un altro indice (semplice) per ovviare al problema. È molto utile in contesti sharded (paragrafo 3.1.4). Non è molto adatto ad essere utilizzato su campi floating-point in quanto tronca i valori a 64 bit. Ciò può creare collisioni.

Gli indici possono avere diverse proprietà.

**Unique:** un campo indicizzato unique fa si che ogni documento inserito, con valori duplicati, venga rifiutato. Molto simile al concetto di chiave primaria dei classici database relazionali. Se un indice composto è definito unico, il vincolo si applicherà alla combinazione dei campi, non al singolo. Questa proprietà ha comportamenti peculiari in determinate situazioni. Un sottodocumento con un campo unique può essere presente più volte all'interno del documento "genitore". Finchè il valore non è duplicato nella

collezione, non sussistono errori. Se un documento non ha il campo indicizzato, verrà aggiunto e salvato con valore “null”. Come per gli alti valori, "null" può comparire solo una volta in una collezione e ciò impedisce di inserire più di un documento senza il campo unico.

**Sparse:** Gli indici sparsi contengono dati riguardanti solo i documenti che hanno tale campo; anche se il suo valore è null. Ogni documento privo di esso non viene considerato. Se un indice di questo tipo produce result set incompleti per determinate operazioni, MongoDB non lo usa, salvo esplicitamente richiesto. Indici composti e sparsi referenziano tutti i documenti che posseggono almeno un campo tra quelli definiti. Le proprietà sparse e unique possono coesistere. Un indice del genere non tollera documenti con valori duplicati per un certo campo, ma ammette tutti quelli che non lo espongono.

**Time-to-Live:** sono speciali indici che MongoDB usa per rimuovere documenti dopo un certo periodo di tempo. È ideale per sistemi di logging o informazioni di sessioni che devono persistere nel database solo per un certo tempo. Ci sono delle limitazioni da rispettare. Un indice composto non può essere “TTL” e il campo su cui è definito deve essere necessariamente di tipo data. In caso di vettore di date, farà fede la più prossima al tempo attuale. Per tutti gli altri aspetti si tratta di normali indici.

### 3.1.3 Replicazione

Lo scopo della replicazione è creare ridondanza e aumentare la disponibilità dei dati. Con multiple copie su diversi server, protegge il sistema in caso di collasso di un singolo server. Permette il recupero in caso di problemi hardware o interruzioni di servizio. Con dati ridondati è possibile dedicarne una copia al disaster recovery, alla reportistica o al

backup.

In alcuni casi la replicazione aumenta l'efficienza delle letture. I client hanno la facoltà di leggere e scrivere da diversi server. Mantenere copie dei dati in diverse località geografiche aumenta la availability per applicazioni distribuite.

Dal punto di vista di MongoDB, la replicazione è un gruppo di istanze che ospitano lo stesso insieme di dati. Ogni istanza assume un ruolo che ne definisce le funzioni. La figura 3.4 mostra l'archetipo di architettura di un replica set.

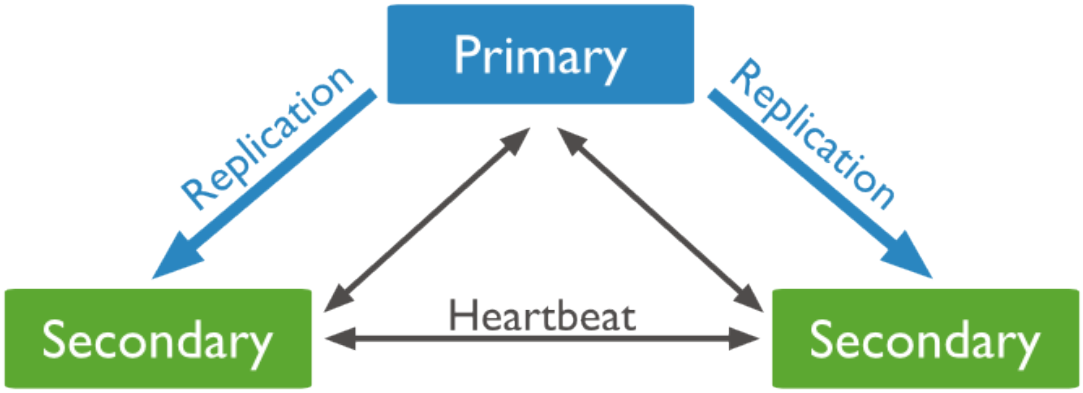


*Figura 3.4 : Modello di un "replica set". Segue il concetto di master-slave.*

Un istanza è detta primaria quando accetta operazioni di scrittura dai client. Per ogni insieme di replicazione può esistere una sola primary. Dato che solo un membro può scrivere dati, i replica set forniscono una strict consistency[10] per tutte le letture effettuate dall'istanza primaria. Per consentire la replicazione dei dati vengono riportati tutti i cambiamenti del data set della primary.

Le istanze cosiddette secondarie, utilizzando i file di log della primaria, duplicano le operazioni sul proprio insieme dei dati. In questo modo

10 La *strict consistency* è un modello di consistenza che forza un sistema ad eseguire le operazioni richieste, esattamente nell'ordine in cui sono state richieste.

riflettono l'insieme primario. Per definizione i client leggono il data set primario ma, come già detto, è possibile esprimere una preferenza per un secondario. Non c'è però sicurezza che i dati reperiti dalle secondary riflettano fedelmente la primary.

I membri di un insieme di replicazione si inviano reciprocamente dei messaggi allo scopo di monitorare lo stato di salute dell'insieme. Se un messaggio non riceve risposta entro 10 secondi, l'istanza che non ha risposto viene marchiata come “inaccessibile”. Se l'istanza inaccessibile è la primaria, scatta un processo di elezione che definirà la sostituta. Le istanze rimanenti votano per decidere quale dovrà assumere il ruolo di primaria. Ci sono alcune condizioni che incidono sull'elezione.

- **Priorità:** è possibile impostare una priorità da zero a mille per i membri. In generale l'istanza con priorità più alta ha più possibilità di essere eletta.
- **Optime:** è il timestamp dell'ultima operazione del log primario eseguita da un membro. Esso non può diventare primario a meno che non abbia il più alto (più recente) optime di tutti gli altri membri visibili.

In caso di un numero pari di istanze in un gruppo, è possibile definire un membro aggiuntivo chiamato “arbitro”. Un arbitro non mantiene alcun dato ma partecipa alle votazioni per evitare situazioni di stallo. Questa istanza non può cambiare funzione; un arbitro rimane sempre un arbitro.

Una secondary può essere configurata per avere particolari comportamenti all'interno del replica set. Si può impedire che diventi primary impostando al sua priorità a zero o può essere inibita la lettura da essa. In questo modo è possibile eseguire applicazioni che richiedono

separazione dal normale traffico. Infine impostando un ritardo nella replicazione è possibile creare snapshot dei dati per il recupero in caso di bisogno. Un membro "delayed" deve avere priorità zero e dovrebbe essere protetto dalla lettura.

### 3.1.4 Sharding

Le basi di dati con un grosso volume di dati e applicazioni ad alto throughput possono mettere in difficoltà le capacità di una singola macchina. Per risolvere questo problema, fondamentalmente esistono due approcci: scalare in verticale o in orizzontale.

Il vertical scaling consiste semplicemente nell'aumentare la potenza hardware di una macchina. Questo però, nei sistemi ad alte performance, ha un costo sproporzionato rispetto ai vantaggi. In conclusione esiste un limite pratico a questo approccio.

Lo sharding o scalamento orizzontale, invece, divide l'insieme dei dati su più server o shards. Ogni server è un database a se stante e insieme, costituiscono un singolo database logico.

MongoDB può scalare in orizzontale definendo uno sharded cluster. Come si vede in figura 3.5, la costruzione di cluster prevede una serie di componenti.

**Shards:** si occupano di immagazzinare i dati, possono essere istanze singole o insiemi di replicazione.

**Query Routers:** sono le interfacce esposte ai client e dirigono le operazioni verso shards opportuni. Un cluster può averne più di uno per bilanciare il carico di richieste.

**Config Servers:** mantengono i metadati dell'architettura. Questi ultimi contengono i dati relativi alla distribuzione del data set nei vari frammenti.

I cluster di produzione hanno esattamente tre server di configurazione.

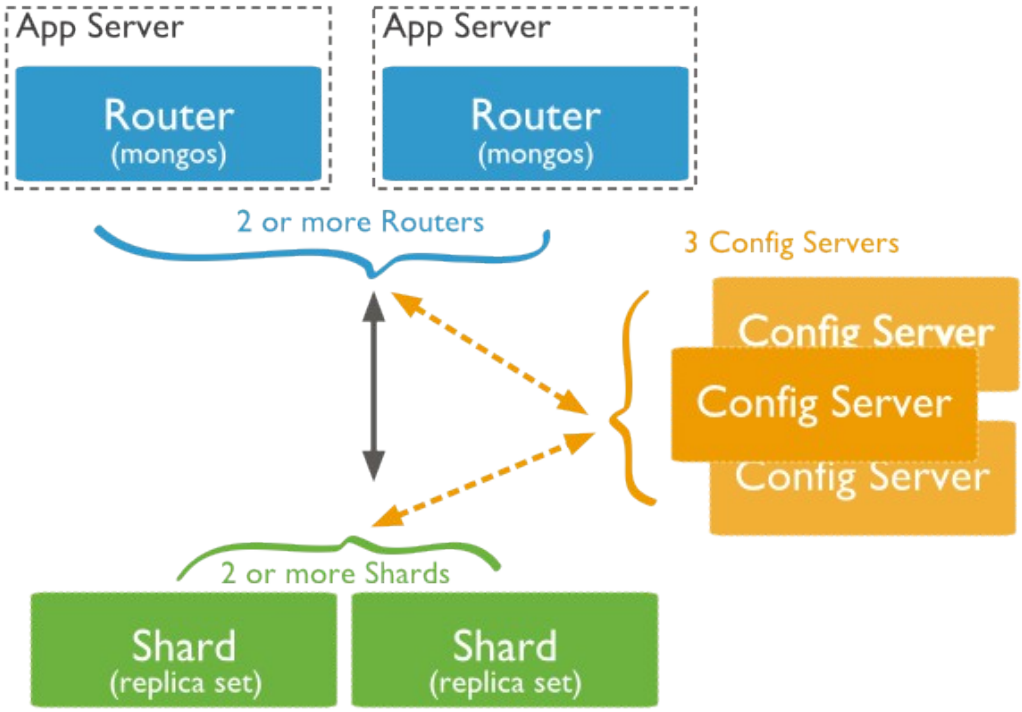


*Figura 3.5: Architettura di produzione di uno "sharded cluster".*

MongoDB distribuisce i dati a livello di collezione e ogni frammento di collezione è identificato da una shard key. Questa chiave è un indice semplice o composto i cui campi esistono in tutti i documenti. MongoDB ripartisce i chunk di dati equamente tra i nodi. Per farlo utilizza due metodi.

**Range Sharding:** I documenti sono divisi in intervalli non sovrapposti secondo il valore della shard key. In questa organizzazione i documenti con chiavi "vicine" sono generalmente nello stesso spezzone di dati e quindi nello stesso shard. L'indice utilizzato in questo caso non può essere multichiave.

**Hash Sharding:** facendo uso di un indice hash, documenti sono suddivisi in bucket corrispondenti alla chiave calcolata. È bene scegliere un campo, su cui calcolare la funzione hash, che abbia una buona cardinalità o

un gran numero di valori distinti. Questo tipo di shard key si comporta bene con valori che crescono monotonicamente. Alcuni esempi sono gli ObjectId o i timestamp.

La scelta del metodo di suddivisione è importante. Ognuno di essi è più performante in certi casi e carente in altri. La range based partition supporta interrogazioni di intervallo molto efficienti. Il query router in questo caso è in grado di identificare gli shard dove risiedono i dati. D'altro canto la distribuzione dei dati potrebbe risultare sbilanciata. Si ponga ad esempio una chiave definita su un campo indicante una informazione come il tempo. In questa situazione una piccola parte degli shard dovrebbe gestire la maggioranza delle richieste, lasciando inattivi gli altri. Al contrario la hash based partition assicura il bilanciamento del carico a spese di efficienti query di intervallo. I valori hash calcolati creano un distribuzione casuale dei documenti. Ciò probabilmente obbligherà una interrogazione ranged a consultare tutti i nodi.

MongoDB permette di personalizzare la politica di bilanciamento usando dei tag. Essi sono associati ad un sottoinsieme di chiavi distribuite e ad uno shard. In questo modo si impone che quel dato insieme sia mantenuto in quello specifico shard. I possibili casi d'uso di questa pratica sono:

- isolare una parte dei dati.
- assicurare che i dati più rilevanti risiedano in shard geograficamente vicini ai client (localizzazione dei dati).

L'aggiunta di dati o nuovi nodi può creare una distibuzione sbilanciata nel cluster. Attraverso due processi in background, MongoDB assicura una partizione equa.

**Splitting:** impedisce ai nodi di aumentare troppo di volume. Quando un

frammento di dati in esso contenuto eccede la dimensione massima specificata[11], viene diviso. Questo comportamento si attiva con inserimenti e modifiche ai documenti. Lo splitting non prevede alcuna migrazione di dati tra gli shard. Frammenti di dati rappresentati da una singola shard key sono indivisibili. Per questo la scelta della chiave di partizione è importante.

**Balancing:** quando un nodo ha troppi frammenti di dati rispetto agli altri, viene lanciata una procedura di bilanciamento. Questo processo sposta porzioni di dati da uno shard che ne ha troppe ad uno che ne ha meno. La migrazione di dati porta con se dell' overhead in termini di banda e carico di lavoro. Per minimizzare l'impatto sulle performance il bilanciatore sposta un solo chunk di dati per volta. Un altro metodo per impattare il meno possibile sulle performance è aspettare fino a che sia raggiunta la "soglia di migrazione". Questa soglia è definita come la differenza tra il numero di "frammenti dati" del nodo che ne ha di più e quello che ne ha di meno. La tabella 3.1 mostra diverse soglie a seconda del numero di frammenti totali.

| Numero di chunk | Soglia di migrazione |
|---|---|
| meno di 20 | 2 |
| 20-79 | 4 |
| più di 79 | 8 |

Tabella 3.1 : Soglie di migrazione rispetto al numero di chunks.

Una volta partito un giro di bilanciamento, esso si ferma solo quando la differenza in frammenti tra due nodi qualsiasi è meno di due, oppure in caso di fallimento di una migrazione.

11 Per default, la dimensione massima è di 64 megabyte.

Aggiungere uno shard crea sbilanciamento e serve tempo prima che l'intero cluster si bilanci. Anche in caso di rimozione di un nodo, è necessario aspettare che i dati migrino su altri, prima di poterlo eliminare.

### 3.1.5 Accesso ai dati

MongoDB, come ogni sistema di salvataggio di dati, espone un interfaccia per l'accesso ai dati. Pur essendo, nei concetti, simile a SQL, la sua sintassi è assolutamente diversa.

**Lettura:** una query di lettura semplice supporta operazioni di selezione e proiezione e produce un cursore. Esistono modificatori che alterano il result set del cursore. Ordinamenti e limiti nel numero di risultati fanno parte di questi modificatori. Un cursore non è isolato durante la sua vita, operazioni di scrittura possono indurre il cursore a mostrare lo stesso documento più volte. Tutte le interrogazioni coinvolgono una sola collezione. L'uso di DBRef (paragrafo 3.1.1) richiede una lettura per ogni collezione coinvolta e l'ordinamento dei risultati non è definito se non esplicitamente richiesto.

**Inserimento:** un nuovo documento deve obbligatoriamente contenere un campo identificativo. Se ne è privo, un campo chiamato "_id" di tipo ObjectId verrà aggiunto e questo valore deve essere unico all'interno della collezione.

**Modifica:** un update coinvolge sempre un solo documento. È comunque presente un opzione per lanciare aggiornamenti multipli. Con l'opzione upsert si comanda a MongoDB di creare il documento se non esiste. I criteri di ricerca in fase di modifica sono gli stessi delle query di lettura.

**Cancellazione:** utilizza i soliti criteri di ricerca per eliminare determinati documenti. Vengono eliminati tutti i documenti corrispondenti, ma è possibile impostare un numero massimo.

Ogni scrittura è isolata, quindi applicazioni concorrenti non presentano particolari problemi. Le operazioni multiple, al contrario, non sono isolate perché non atomiche.

### 3.1.6 Aggregation framework

Si tratta di una funzionalità speciale di MongoDB per l'aggregazione dei dati. Eseguire aggregazioni di questo tipo semplifica molto il codice applicativo e limita l'uso di risorse. Il framework è modellato sul concetto di "pipeline multi-stadio" in cui i documenti entrano e vengono trasformati in un aggregazione degli stessi. Un operazione di aggregazione prende in input un intera collezione e prevede la definizione delle "fasi di aggregazione" che comporranno la pipeline. Le fasi disponibili richiamano le operazioni classiche SQL:

- Proiezione: modifica il documento in input rimuovendo o aggiungendo campi.
- Selezione: equivalente della clausola "where" SQL.
- Ordinamento
- Raggruppamento: equivalente della clausola "group" SQL.
- Limit: esclude gli ultimi "n" documenti.
- Skip: esclude i primi "n" documenti.

Per aumentare le performance del framework è buona norma porre le fasi di selezione, limit e skip all'inizio della pipeline in modo da ridurre il più possibile la mole di dati per le altre fasi. Come le altre operazioni di lettura, l'aggregazione, trae vantaggio dalla definizione di indici sulla collezione.

## 3.2 Java Messaging Service

JMS è l'insieme di API, appartenente Java EE[12], che consente ad applicazioni presenti in una rete di scambiarsi messaggi tra loro. È definito dalle specifiche sviluppate sotto la Java Community Process[13] come JSR 914 [Jms03].

Il suo utilizzo è richiesto a quelle applicazioni che necessitano di:

- Disaccoppiamento: i moduli che utilizzano questa tecnologia non hanno bisogno di esporre interfacce, è sufficiente che inviino o ricevano i propri messaggi dalla coda JMS.
- Asincronia: una coda JMS permette di inviare messaggi senza dover aspettare la risposta.
- Affidabilità JMS assicura che ogni messaggio sia spedito una e una sola volta. Altri livelli di affidabilità (messaggi duplicati o persi) sono comunque permessi.

I componenti che costituiscono l' architettura di questa tecnologia sono:

- JMS Provider: un sistema che implementa l'interfaccia JMS e fornisce strumenti di amministrazione e controllo.
- JMS Client: un componente che invia e riceve messaggi attraverso il provider.
- Messaggi: oggetti usati per la comunicazione tra i client.

Le specifiche JMS prevedono due metodi di comunicazione, utili in diversi scenari applicativi.

12 Java Enterprise Enviroment
13 Meccanismo con lo scopo di sviluppare specifiche tecniche standard per Java.

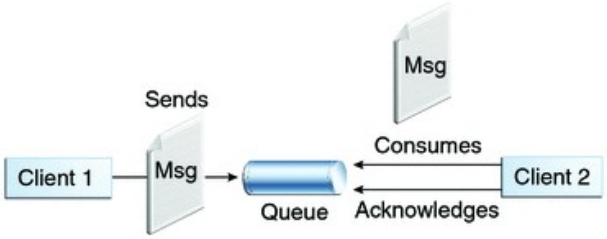


*Figura 3.6 : Schema di messaggistica PTP.*

**Point-to-point:** la figura 3.6 mostra l'idea alla base di questo metodo, Basato sul concetto di “code”, “mittenti” e “destinatari”. In questo contesto possono esistere diversi mittenti ma solo un destinatario. I messaggi sono recapitati ad una coda gestita dal provider. Il destinatario può recuperare un messaggio indipendentemente da quando è stato spedito. Infine notifica la ricezione con successo del messaggio. Questo metodo è utile quando i messaggi devono essere ricevuti sempre dallo stesso destinatario.


*Figura 3.7 : Schema di messaggistica publish/subscribe*

**Publish/subscribe:** in figura 3.7 è mostrato il secondo metodo. Esso si fonda sul concetto di “topic”. Un topic è un contenitore di messaggi simili alle code. Differiscono da queste ultime nel fatto che i messaggi non sono eliminati appena recapitati al destinatario. Questo fa si che ogni messaggio possa essere processato da più destinatari. Per poterli ricevere, un destinatario deve “iscriversi” ad un certo topic. Il contesto più adatto per l'uso di questo metodo è quando è necessario mantenere memorizzati messaggi affinché molteplici destinatari possano riceverli in tempi diversi.

I messaggi sono intrinsecamente asincroni. È sufficiente definire un listener di messaggi. Quando un messaggio arriva alla coda o al topic, il provider, richiama il listener che processerà il messaggio. La sincronia dei messaggi è possibile, non ci sono limiti di tempo tra la produzione e il consumo. Il destinatario deve comunque richiedere i messaggi a lui indirizzati in modo esplicito.

ActiveMQ [Act15] è l' implementazione di un JMS provider gestita Apache. È il più famoso e potente server open-source di messaggistica. Supporta molti protocolli di comunicazione e diversi linguaggi. Implementa completamente la specifica JMS 1.1 [Jms03] e J2EE 1.4 [J2e04].

## 3.3 Architetture software

Il paragrafo che segue illustra le architetture software più utilizzate al momento, ponendo particolar attenzione su quelle utilizzate sia nell'architettura EasyMarket che nel nuovo componente.

### 3.3.1 Architettura a tre livelli

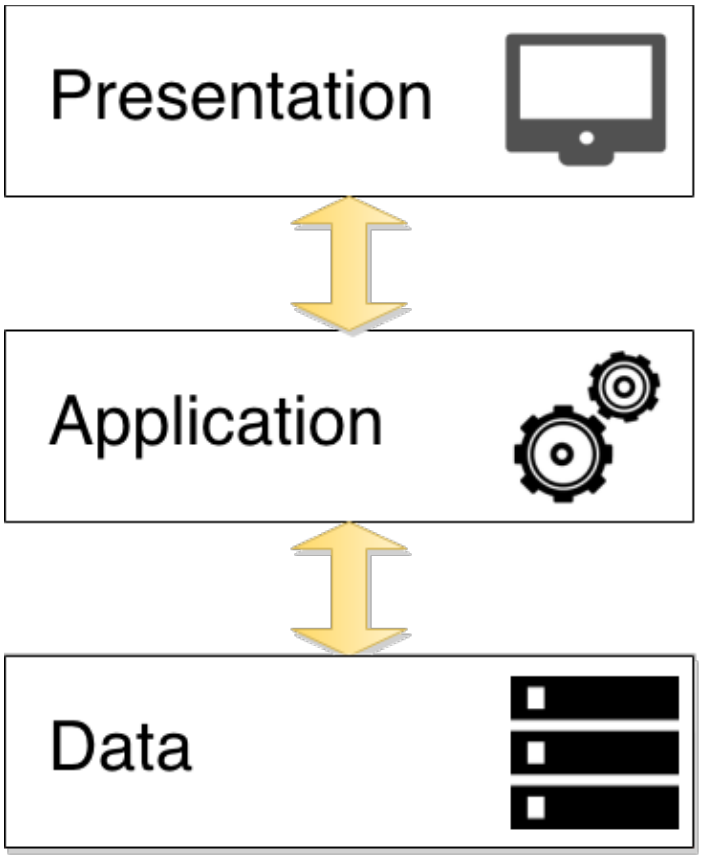


*Figura 3.8 : I tre strati logici di un architettura a tre livelli*

Nell'ingegneria del software, un architettura multi-livello presenta una struttura in cui le sue fuzionalità sono separate in strati logici (figura 3.8). Questi strati sono in comunicazione diretta tra loro, ogni livello richiede e offre servizi ai livelli adiacenti. Al giorno d'oggi viene spesso utilizzata un architettura composta da tre strati.

- **Presentazione:** è il livello più alto dell'applicazione. Il suo compito è

mostrare le infomazioni in un modo comprensibile all'utente.

- **Applicazione:** si tratta di uno strato in cui viene inserita tutta la logica di business. Non contiene nessuna funzionalità di presentazione o accesso ai dati.
- **Dati:** è lo strato più basso e il suo unico scopo è permettere al livello di applicazione di reperire e memorizzare i dati nel database sottostante.

La maggior parte dei componenti descritti nell'architettura EasyMarket si basa su questo tipo di architetture.

### 3.3.2 Representational State Transfer

Il termine REST fu usato la prima volta nel 2000, nella tesi di dottorato di Roy Fielding [Fie00]. È un architettura pensata per applicazioni connesse in rete e si basa sul concetto di risorse. Una risorsa è una fonte di informazione raggiungibile attraverso un URI. I sistemi RESTful seguono alcuni principi di progettazione.

- **Client-server:** il client e il server sono separati. Ciò significa che entrambi non conoscono i dettagli implementativi dell'altro. In questo modo si aumenta la modularità di un architettura.
- **Stateless:** Ogni risposta contiene le informazioni necessarie a completare la richiesta. Non vengono mantenuti dati di sessione.
- **Interfaccia uniforme:** la comunicazione deve avvenire attraverso un interfaccia indipendente dalle implementazioni di client e server. I messaggi scambiati devono essere auto-descrittivi e in un formato standard.
- **Cacheable:** deve essere definito se le informazioni sono "cacheabili"

o no. Questo previene l'uso di dati inappropriati o obsoleti e diminuisce l'interazione tra le parti.

- **Stratificazione:** Un client non può sapere se è connesso direttamente al server o ad un intermediario. Gli intermediari consentono l'uso di funzionalità come il bilanciamento di carico e l'autenticazione.

Nel caso di applicazioni basate sul web, REST è implementato usando il protocollo HTTP. Le richieste avvengono attraverso i metodi GET, POST, PUT, DELETE. Le risorse sono identificate da un URL e il metodo utilizzato definisce l'operazione da eseguire. Il corpo dei messaggi di richiesta e di risposta è codificato in formati standard come XML e Json. La figura 3.9 mostra alcuni esempi di risorse REST e le operazioni eseguibili su di esse.

| example.com/resources<br>collezione di risorse | | | |
|---|---|---|---|
| **GET** | **PUT** | **POST** | **DELETE** |
| visualizza la lista | rimpiazza la collezione con un altra | aggiungi un nuovo elemento | cancella la collezione |

| example.com/resources/item/17<br>risorsa singola | | | |
|---|---|---|---|
| **GET** | **PUT** | **POST** | **DELETE** |
| visualizza l'elemento | rimpiazza l'elemento nella collezione | generalmente non usato | cancella l'elemento dalla collezione |

*Figura 3.9 : Esempi di risorse REST*

# Capitolo 4

# BiAnalyzer

BiAnalyzer è il nome del nuovo componente sviluppato per EasyMarket. Nel corso di questo ultimo capitolo saranno analizzate tutte le fasi del suo sviluppo. Si esporranno i problemi che l'azienda intende risolvere grazie a questo componente. Essi verranno poi analizzati per estrarre delle specifiche ben definite. Seguirà una fase di progettazione in cui si descriverà l'architettura del nuovo componente. Infine sarà mostrata la prima integrazione di BiAnalyzer, corredata di test di collaudo.

## 4.1 Specifiche

EasyMarket come già descritto nel paragrafo 1.3, possiede un architettura informatica molto modulare e varia. L'azienda usa già alcuni strumenti per controllare l'attività dei suoi servizi. Questi però sono spesso piccoli applicativi costruiti per uno scopo ben preciso e specializzato. L'azienda ha così espresso il desiderio di creare un nuovo modulo da integrare nel suo sistema. Consiste fondamentalmente in una banca dati integrata dove poter salvare e leggere dati.

L'utilizzo che verrà fatto dei suddetti dati sarà molto vario, ad esempio:

- Business Intelligence: memorizzazione di informazioni sull'andamento dell'azienda e dei suoi servizi.
- Indagini di mercato.
- Merchandising: costruzioni di banner pubblicitari volti a

massimizzare le probabilità di prenotazione.

È doveroso fare una precisazione riguardo agli utilizzi per scopi promozionali. Come già spiegato in precedenza i propri prodotti turistici sono acquisiti da fornitori esterni. La prima impressione suggerirebbe di reperire le informazioni come prezzi e offerte direttamente alla fonte. Tuttavia esistono dei vincoli che scoraggiano e/o impediscono questa soluzione.

**Look-to-Book:** è una misura utilizzata nell'industria del turismo online che mostra la proporzione tra ricerche (visite) e prenotazioni. I fornitori di EasyMarket vincolano a rispettare un certo valore, pena il pagamento di una penale per ogni ricerca fuori parametro. Lanciare ricerche non finalizzate a prenotazione aumenta la probabilità di sforare questa quota.

**Dati non attendibili:** i tour operator online, compresi i fornitori di EasyMarket, fanno largo uso di sistemi di caching per vari scopi, tra cui quelli appena descritti. Non è quindi possibile determinare se un dato è reale o solo una copia in cache, vanificando lo sforzo di richiedere sempre dati aggiornati. Inoltre si fa notare che l'utilizzo dei dati aggiornati è auspicabile ma non indispensabile, e comunque di priorità minore rispetto al tempo richiesto per la presentazione dei dati all'utente.

Per questi motivi il nuovo componente conterrà le informazioni salvate anzitempo relative a questi casi d'uso. Naturalmente dovranno essere previsti adeguati metodi di aggiornamento.

BiAnalyzer è usato come una banca dati centralizzata. Deve accogliere i dati proveniente da qualsiasi altro modulo nell'architettura. Questi ultimi devono poter salvare qualsiasi informazione ritengano opportuna. Inoltre bisogna prevedere la possibilità che alcuni flussi informativi richiedano

alcune elaborazioni extra all'applicazione. Calcoli statistici, confronti e trasformazioni dei dati sono esempi delle suddette elaborazioni.

Infine la logica implementata deve avere un impatto minimo sull'esecuzione degli altri moduli. La costruzione, il salvataggio e il recupero dell'informazione deve essere il più veloce possibile e assolutamente non bloccante per gli altri componenti.

Nella prima integrazione, il modulo salva le informazioni principali riguardanti il ciclo di vita della prenotazione di un hotel. Sono salvate ricerche, preventivi, prenotazioni e cancellazioni. Gli eventi contengono dettagli sull'utente, nonché i parametri di ricerca (dove presenti). Sono memorizzate anche le offerte che l'utente andrà a visualizzare. Nel caso di preventivi, prenotazioni e cancellazioni, il prezzo di un offerta è salvato in tutte le sue componenti.

Nel caso delle ricerche, vengono effettuate elaborazioni aggiuntive. Per ogni hotel è calcolato prezzo minimo, medio e massimo tra tutte le offerte. Queste ultime vanno poi conteggiate divise per fornitore, account del fornitore, e sotto-fornitore (se presente). La ricerca nel suo complesso contiene il prezzo minimo, medio e massimo tra tutti gli hotel, divisi per categoria di stelle.

## 4.2 Analisi dei requisiti

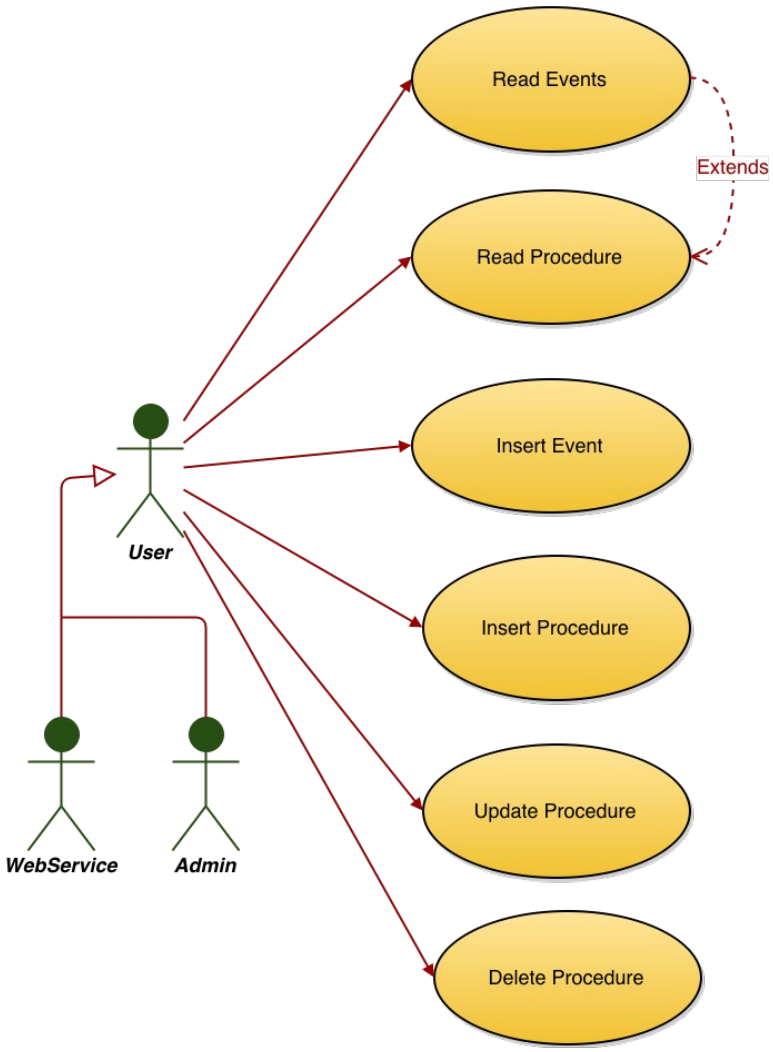


*Figura 4.1 : Diagramma dei casi d'uso di BiAnalyzer.*

Dalle specifiche appena elencate è possibile estrapolare alcune caratteristiche che BiAnalyzer deve avere.

**Efficienza:** in questo caso efficienza equivale a throughput. Il throughput è un indicatore delle prestazioni che tiene conto del numero di operazioni compiute in un lasso di tempo definito. Nel caso specifico le operazioni possibili sono le quattro funzioni classiche Create, Read, Update, Delete, in altre parole, lettura e scrittura.

**Flessibilità:** il modulo non ha controllo sui dati che salva. Spetta al client decidere la struttura e la semantica delle informazioni. Il modulo deve essere quindi flessibile e accettare dati molto eterogenei e variabili.

**Manutenibilità:** in virtù della flessibilità che BiAnalyzer deve avere, serve anche che sia facilmente mantenibile. Non è possibile creare un applicazione che risponda perfettamente in qualsiasi caso. Nei casi in cui questa flessibilità fallisca deve essere possibile intervenire con operazioni di manutenzione poco invasive e complicate.

**Interoperabilità:** l'ambiente in cui il modulo andrà ad integrarsi, come si è visto, contiene componenti che svolgono funzioni e usano tecnologie molto diverse tra loro. Il modulo deve garantire il servizio a tutti, indipendentemente. Per questo il modulo dovrà essere capace di comunicare con gli altri sistemi con affidabilità e semplicità.

Come ogni base di dati, il modulo usa dei criteri di ricerca per reperire le giuste informazioni. Quindi deve essere definito un metodo di interrogazione fruibile dall'utente. La figura 4.1 mostra l'interazione degli utenti con il nuovo sistema. In particolare si nota che la lettura dei dati prevede l'esecuzione di un interrogazione precedentemente salvata.

## 4.3 Architettura

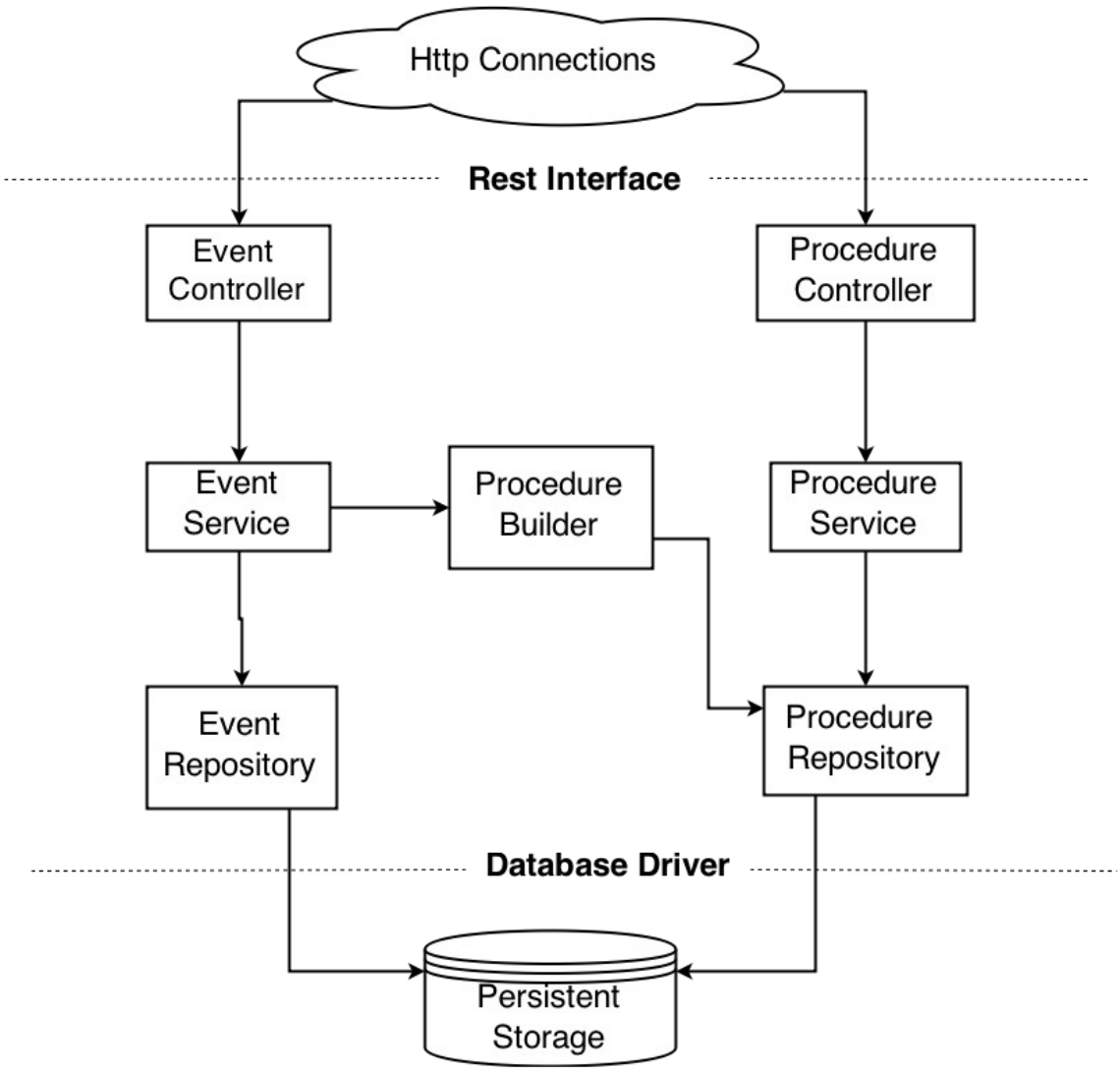


*Figura 4.2 : Architettura di BiAnalyzer*

In questo paragrafo sarà descritta l'achitettura che ha permesso di soddisfare i requisiti definiti nelle pagine precedenti.

La figura 4.2 mostra uno schema concettuale che aiuta a capire la struttura del nuovo modulo. Rimanendo fedeli alla filosofia generale dell'architettura EasyMarket, l'interfaccia di comunicazione è stata implementata come un servizio di tipo REST, già introdotto in precedenza.

Nello schema si possono notare due flussi di controllo paralleli. Uno di

essi è quello principale che si occupa di gestire le richieste dei client. Il secondo flusso serve a gestire un sistema di stored procedure. Permette il salvataggio e la modifica di interrogazioni predefinite che saranno poi utilizzate nelle operazioni di lettura riguardanti i dati principali. Le procedure salvate sono dei modelli che all'occorrenza sono popolati con i dati ricevuti in input. Il componente "Procedure Builder" si occupa proprio di questo. Esso recupera il modello di query richiesto e utilizzando i parametri passati in input crea un'interrogazione eseguibile sul database sottostante.

Il motivo per cui è stato implementato questo sistema è che la base di dati di BiAnalyzer non può essere un database relazionale. Se così fosse sarebbe bastato utilizzare i sistemi già esistenti per creare query dinamiche. I requisiti di flessibilità e manutenibilità spiegati in precedenza scoraggiano l'utilizzo di database convenzionali. Come soluzione alternativa è stato adottato un database NoSQL, MongoDB. Quest'ultimo non possiede alcun sistema di stored procedure e l'opzione di definire staticamente le query in fase di implentazione non era accettabile. Per questo è stato progettato il gestore di procedure spiegato prima.

Sempre in riferimento alla figura 4.2, si può riconoscere un' architettura a tre livelli. La natura di quest'ultima è descritta nel capitolo precedente. Al livello più alto stanno i controller. Sono i componenti in cui è definita l'interfaccia REST. Oltre a ricevere le richieste si occupano anche di segnalare eventuali errori.

I "service" sottostanti hanno il compito di contenere la logica applicativa di tutto il modulo. Il servizio nel flusso di controllo degli eventi svolge diverse funzioni. Le elaborazioni extra elencate nelle specifiche saranno effettuate qui. Il componente, prima di salvare le informazioni, le userà per

derivare i dati aggiuntivi richiesti. Inoltre ha il compito di comunicare con il costruttore delle interrogazioni. È compito suo passare i parametri dinamici da inserire nelle query. Dopo di che invia l'interrogazione che gli viene restituita al livello sottostante, per essere eseguita.

Tra controller e service degli eventi è implementata un'importante funzionalità. Questo è il punto in cui l'inserimento di nuovi eventi è reso asincrono. Il controller non passa direttamente l'evento al service, ma lo deposita su una coda di messaggi. La tecnologia usata in questo caso è quella spiegata nel capito precedente, Java Message Service. Il service in questo caso funge da destinatario, che infatti implementa un listener di messaggi. Con questa funzionalità le operazioni di inserimento risulteranno estremamente efficienti e non bloccanti, come da specifica.

Questo approccio asincrono però ha i suoi contro. Non sarà possibile notificare al client l'esito di un inserimento, o comunque non in modo dettagliato. A causa di ciò è indispensabile implementare un sistema robusto, che gestisca il più possibile gli errori, e un sistema di logging molto capillare per dettagliare ogni possibile caso anomalo.

In fase di lettura, ovviamente le richieste non possono essere asincrone, per cui le prestazioni sono dipendenti dalla rapidità di reperimento dei dati, su cui bisogna porre particolare attenzione.

Nell'ultimo strato dell'architettura sono presenti i componenti che collegano l'applicazione al database. Come già detto ,il sistema si basa su MongoDB e i “repository” contengono i metodi per interfacciarsi con quest'ultimo.

## 4.4 Progettazione e implementazione

In questo paragrafo verrà mostrata la reale implementazione del modulo

e le soluzioni adottate per venire incontro ai requisiti delle specifiche.

Tutto il progetto utilizza il framework di Spring e suoi moduli spiegati in precedenza. In particolare sono stati usati:

- **Spring-web:** contentente tutti gli oggetti necessari a creare l'interfaccia REST e a definire le modalità con cui processare le richieste.
- **Spring-jms:** l'implementazione di Spring delle specifiche JMS elencate precedentemente.
- **Spring-data-mongodb:** come spiegato nel primo capitolo, è un interfaccia di comunicazione con MongoDB costruita sfruttando il driver nativo per Java fornito dal database stesso.

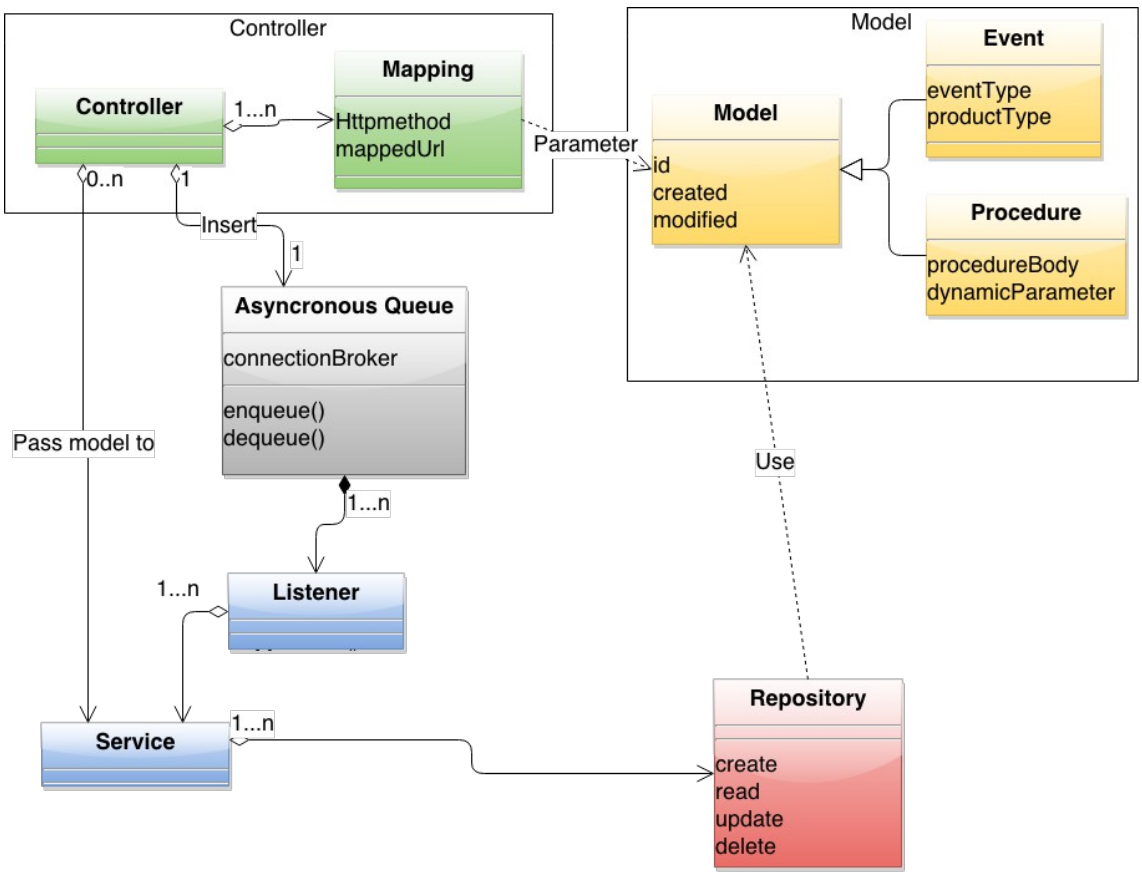


*Figura 4.3 : Classi di progettazione di BiAnalyzer*

Il JMS provider utilizzato è ActiveMQ [Act15], il quale è stato introdotto nel capitolo precedente. È utilizzato in modalità "point-to-point", in quanto non c'è necessità di definire più destinatari.

La figura 4.3 mostra un diagramma delle classi utile a capire la reale struttura di BiAnalyzer. Questa modellazione è valida per entrambi i flussi di controllo spiegati in fasi di analisi. L'unica differenza è che le procedure non sono inserite utilizzando la coda asincrona. È stato omesso il componente responsabile della costruzione delle interrogazioni, perchè verrà mostrato più nel dettaglio successivamente.

### 4.4.1 Flessibilità attraverso polimorfismo

Il paragrafo che segue illustrerà un problema progettuale abbastanza complicato e la soluzione adottata in questo frangente per risolverlo.

Come espressamente richiesto nelle specifiche, il modulo deve essere flessibile rispetto alle strutture dati. Queste ,d'altro canto, per esser utilizzabili da BiAnalyzer e dai suoi client, devono avere un minimo di struttura. Informazioni completamente dinamiche richiederebbero una fase di conversione, per creare oggetti con campi definiti.

La soluzione implementata utilizza un costrutto tipico dei linguaggi orientati agli oggetti. Grazie al polimorfismo BiAnalyzer è in grado di gestire qualsiasi dato strutturato.

La figura 4.4 mostra la gerarchia di classi utilizzata dal modulo e dal web-service degli hotel per comunicare. Un oggetto fa da "radice" e da esso è possibile derivare un determinato evento. "EngineEvent" è la classe che va estesa se si vuole comunicare con BiAnalyzer. I campi "eventType" e "productType" sono utilizzati per discernere il tipo di struttura che si sta utilizzando.

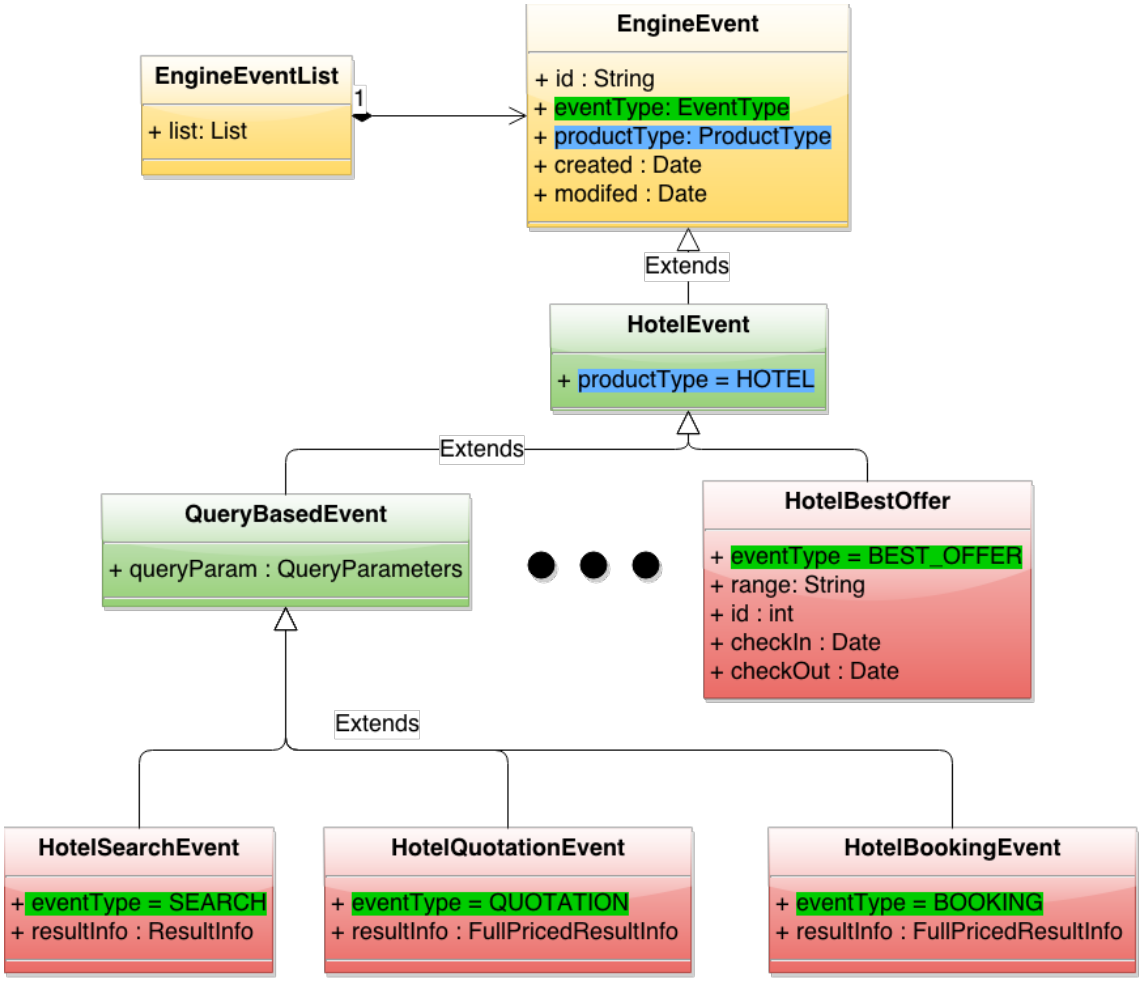


*Figura 4.4 : Gerarchia delle classi utilizzate nella prima release*

Questa soluzione presenta sia vantaggi che svantaggi. MongoDB supporta molto bene il polimorfismo, perciò da questo punto di vista la soluzione è ottima. Durante le operazioni di lettura e scrittura, la reale struttura dei dati è trasparente al modulo. Tuttavia le specifiche richiedono che alcuni dati vengano rielaborati prima del salvataggio. In questi casi è necessario sapere di quale evento si tratta.

A questo scopo il componente "EventService" visto in precedenza è stato suddiviso in sottocomponenti diversi, che implementano un interfaccia comune. Ognuno di loro si occuperà di uno specifico tipo di prodotto fornendo la possibilità di effettuare le elaborazioni suddette. Al bisogno, utilizzando i due campi identificativi, verrà riconosciuto l'evento e di

conseguenza il service opportuno.

Durante la costruzione delle query, per velocizzare la lettura, sono aggiunti due criteri basati sui campi "eventType" e "productType". Definendo un indice su questi valori nel databse, le ricerche saranno molto più selettive e veloci.

Per disaccoppiare il più possibile client e server questo sistema gerarchico di eventi è contenuto in una libreria a parte. Questa conterrà solo le dipendenze strettamente necessarie in modo da non costringere il client ad importare dipendenze software che non usa.

### 4.4.2 Query Template Engine

L'interfaccia di memorizzazione delle interrogazioni, vista nell'architettura di BiAnalyzer, permette l'inserimento di stored procedure "a caldo", cioè senza bisogno di interrompere il servizio. Unita al sistema di costruzione delle query che verrà ora mostrato, garantisce la massima flessibilità e operatività.

MongoDB prevede due tipologie di interrogazioni, come visto in precedenza:

- **Query semplice:** sono interrogazioni classiche, presenti in ogni database. A differenza di SQL però non supporta operatori di aggregazione.
- **Aggregazione:** sfrutta una pipeline multi-stadio per processare un intera collezione in uno o più documenti contenenti dati aggregati.

La figura 4.5 mostra come il componente "Procedure Builder" valorizza i campi dinamici dei "template" delle interrogazioni.

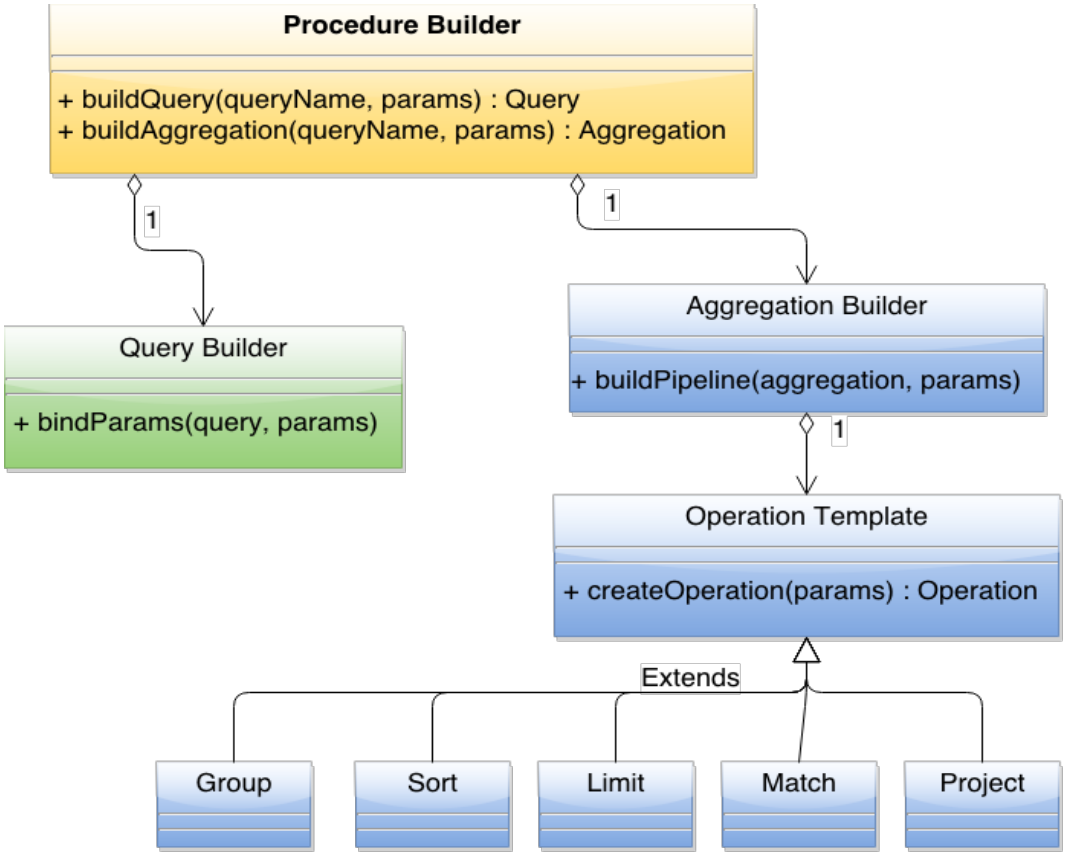


*Figura 4.5 : Classi usate per costruire le interrogazioni*

Le query classiche sono costruite in modo abbastanza semplice. Su MongoDB viene mantenuto un documento contenente il corpo della procedura. Nella query sono definiti dei "segnaposto"; delle stringhe che andranno poi sostituite con il dato reale. Questi segnaposto sono raccolti in una lista che viene salvata insieme al corpo. Il client, quando formulerà la sua richiesta, indicherà i parametri dinamici utilizzando questi segnaposto.

Le query di aggregazione vengono costruite con un processo più complesso. Essendo interrogazioni piuttosto complicate, l'approccio appena descritto non è sufficiente. Un'aggregazione su MongoDB può utilizzare molti parametri, che vengono ripetuti durante i diversi stadi dell'interrogazione. Tenere conto di tutti i casi possibili è molto difficile, sia dal punto di vista progettuale che implementativo. La soluzione adottata prevede di costruire la pipeline di aggregazione usando metodi predefiniti.

Grazie al supporto di Spring per MongoDB, la costruzione di una singola operazione è particolarmente semplice. Il documento su database che mantiene i dati relativi a queste interrogazioni è formato da:

- una lista di segnaposto come avviene per le query semplici.
- una serie di sottodocumenti contenenti informazioni riguardanti il tipo di operazione (raggruppamento, ordinamento, selezione ecc.), il nome del template da utilizzare e gli eventuali segnaposto da utilizzare in quella specifica operazione.

Tutti i modelli di operazione sono contenuti in classi che li raggruppano per tipo (figura 4.5). Grazie al documento appena descritto, i metodi di queste classi possono essere richiamati con i relativi parametri, restituendo una fase della pipeline di aggregazione richiesta.

La soluzione appena mostrata però presenta un difetto. I metodi che restituiscono le operazioni della pipeline sono definiti in fase di scrittura del codice. Nei primi tempi dopo il primo rilascio di BiAnalyzer, probabilmente sarà necessario aggiungere spesso nuovi template. Questo richiederà necessariamente il rilascio di nuove versioni dell'applicativo. Col tempo, progettando modelli il più generici possibile, si dovrebbe ovviare a questo inconveniente, riutilizzando quelli già definiti.

## 4.5 Integrazione

Questo paragrafo mostrerà come il web-service dedicato agli hotel abbia integrato le funzionalità di BiAnalyzer.

Le specifiche, come già detto, richiedono che vengano salvati gli eventi legati a ricerche, preventivi, prenotazioni e cancellazioni. La struttura di alcuni di questi eventi è mostrata nell figura 4.4 del paragrafo precendente.

Le integrazioni sul web-service "hotel" devono essere eseguite seguendo alcune linee guida:

- Configurazione: tutte le proprietà relative ad un modulo integrato devono essere in un file, richiamato a runtime dal web-service. Nel caso di BiAnalyzer, la sua proprietà è l'URL a cui è possibile raggiungerlo. Esiste un copia di qusti file per ogni ambiente di sviluppo (es: test, produzione). I questo modo è possibile modificare alcuni aspetti dell'integrazione a seconda dell'ambiente.
- Interruzione di servizio: EasyMarket ritiene utile avere la possibilità di escludere alcune funzionalità dai web-service, se necessario. Il nuovo modulo rientra proprio tra queste. Nello specifico il web-service hotel mantiene sul suo database dei flag. Questi indicano se una certa integrazione è attiva in un determinato ambiente di sviluppo. Eseguendo controlli su questi "interruttori", il motore hotel può dinamicamente scegliere cosa integrare e cosa no.

Il motore di prenotazione hotel, per usare BiAnalyzer, ha dovuto costruire un client REST. Questo ha il compito di formulare le richieste da inviare al componente. Utilizzando la libreria gerarchica introdotta precendentemente, il client salva e recupera le proprie informazioni. La tecnologia su cui si basa il nuovo client è Jersey. Come già accennato nel primo capitolo è una libreria per servizi RESTful e tutto il web-service hotel la utilizza, perciò la scelta è stata obbligata.

Il motore hotel possiede un oggetto che si occupa di tutto il ciclo di vita di una prenotazione. In esso sono definiti i metodi che gestiscono il cambiamento di stato di un booking. Ognuno di questi metodi è stato modificato per raccogliere i dati che processa e inviarli a BiAnalyzer. Nella

figura 4.6 è mostrato "CommunicationManager", l'oggetto appena descritto e il sistema di creazione e invio degli eventi.

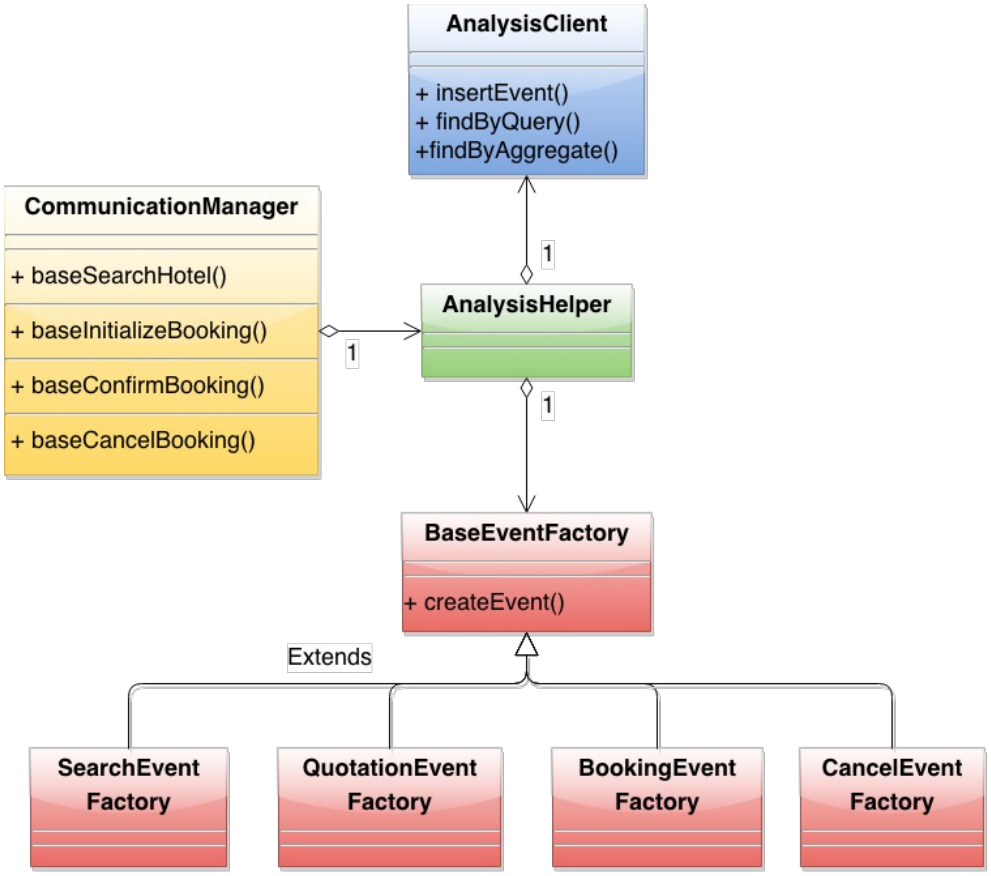


*Figura 4.6 : Sistema di creazione e salvataggio degli eventi*

"AnalysisHelper" contiene dei metodi che inglobano la fase di creazione e invio degli eventi al modulo di analisi. Per costruire la struttura dati adeguata, si fa uso di "fabbriche di eventi". Ognuna di queste classi espone i metodi per costruire un evento sulla base dei dati passati da "CommunicationManager". Gli eventi così creati vengono poi passati al client descritto prima, per il savataggio.

## 4.6 Test

In questo paragrafo verrano mostrati i risultati dei test effettuati per provare la stabilità ed efficienza di BiAnalyzer.

Le prove effettuate, sostanzialmente, erano mirate a testare le principali funzionalità del componente sotto grossi carichi di lavoro. Le richieste effettuate sono di quattro tipi:

1. Richiesta di tutti gli eventi di "prenotazione confermata"
2. Inserimento di un evento
3. Lettura di eventi attraverso interrogazione semplice
4. Lettura di dati utilizzando una query di aggregazione.

Per impostare i test è stato usato un tool apposito, Jmeter [Jme15].Questo strumento permette di costruire batterie di test per molti tipi di applicazioni diverse. Consente di creare pool di thread per l'esecuzione di prove diverse e indipendenti o per incrementare il carico di lavoro sull'applicativo sotto esame.

L'applicazione, durante i test era eseguita nell'ambiente di pre-produzione di EasyMarket, dove sono rilasciate alcune altre applicazioni. Il client era un pc quad-core connesso in wireless all'esterno della rete aziendale.

La tabella 4.1 mostra i risultati di questi test e il risultato complessivo. È stata usata una batteria di otto thread che hanno eseguito le quattro richieste concorrentemente per un totale di centomila prove.

| Test | Prove | Media (ms) | Soglia 90% (ms) | Throughput (richieste/sec) |
|---|---|---|---|---|
| 1 | 25.000 | 177 | 266 | ~5.6/sec |
| 2 | 25.000 | 174 | 220 | ~5.7/sec |
| 3 | 25.000 | 87 | 110 | ~11,5/sec |
| 4 | 25.000 | 89 | 128 | ~11,2/sec |
| Totale | 100.000 | 131 | 209 | ~34/sec |

*Tabella 4.1 : Risultati dello stress test.*

La soglia del 90 percento è un indicatore molto importante. Mostra che il 90 percento delle richieste totali ha ricevuto risposta entro 0,3 secondi. Un throughput di circa 34 risposte al secondo è un ottimo risultato, considerando che l'applicazione eseguiva in un ambiente di prova e non dedicato. Anche il fatto che fosse un unico client a formulare le richieste ha influito negativamente su questo indicatore.

La figura 4.7 mostra l'andamento delo stesso test nell'arco di tempo in cui è stato eseguito.

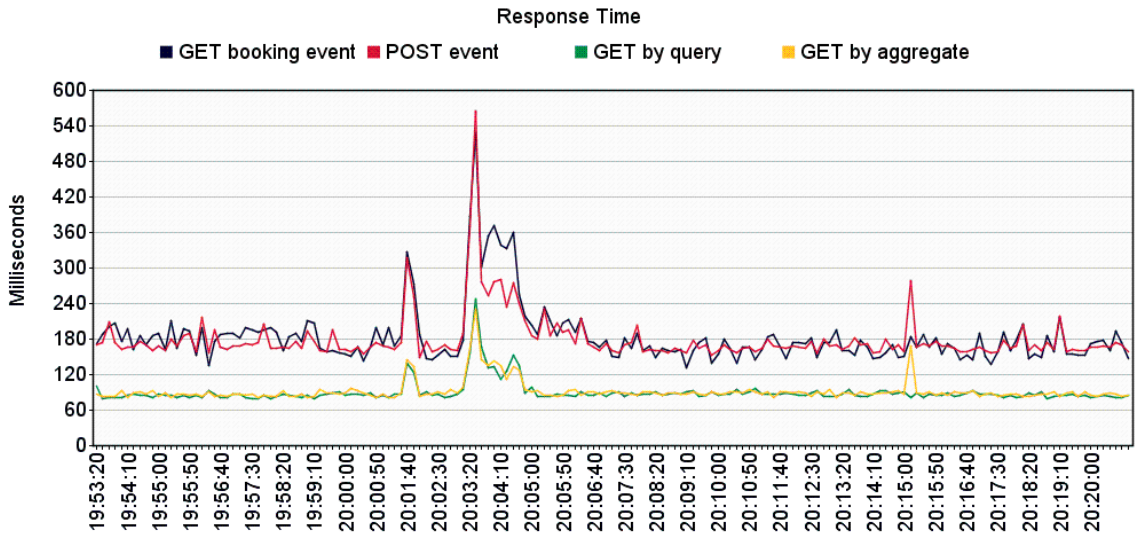


*Figura 4.7 : Grafico temporale dei risultati dei test.*

Come si può notare le prove sono durante per circa mezz'ora. Durante questo intervallo è stato tenuto traccia dei tempi di risposta a step di cinque secondi. Si può vedere come l'applicazione rimanga stabile durante tutto l'arco di tempo. Il picco avvenuto verso le ore 20 è probabilmente dovuto ad un rallentamento del traffico sulla linea. Infatti subito dopo i tempi di riposta si sono ristabilizzati, fino alla fine dei test.

# Conclusioni

In questo lavoro di tesi è stato mostrato lo sviluppo di un software per la raccolta e l'analisi dei dati di un sistema di prenotazione online.

È stato descritto il businness in cui l'azienda committente è inserita e le problematiche che hanno inciso anche sullo sviluppo del nuovo progetto.

Per poter motivare le scelte progettuali descritte durante questa trattazione è stato necessario spiegare a fondo l'architettura in cui il progetto sarebbe stato integrato. Lo scopo ultimo di questo componente è creare una banca dati da cui poter estrapolare dati significativi sull' andamento dell'azienda. Sono state elencate le maggiori tecnologie utilizzate e la filosofia di sviluppo di quest'ultima.

I database non relazionali sono un argomento centrale di questa tesi. È stata illustrata la loro storia e che cambiamenti il movimento NoSQL ha portato nel panorama informatico. È stato introdotto MongoDB, le sue funzionalità principali e le sue potenzialità.

Infine è stato illustrato il percorso effettuato per costruire BiAnalyzer, il nuovo modulo oggetto della tesi. Partendo dalle specifiche, sono stati individuate le qualità più importanti da implementare ed evidenziati i problemi di progettazione da risolvere.

Grazie ai test effettuati post-rilascio, si è potuto dimostrare il grande potenziale di BiAnalyzer. Esso si è dimostrato stabile, veloce e di semplice utilizzo. MongoDB gioca un ruolo di primaria importanza, fornendo un base di dati che si sposa bene con l'adattabilità del resto del modulo.

Il sistema è tuttora in piena evoluzione. Sono già stati considerati diversi casi d'uso da implementare nel prossimo futuro. Le possibilità di utilizzo sono illimitate.

Da parte del sottoscritto, l'implementazione di questa tesi ha portato molti vantaggi. Ho potuto approfondire l'uso di molte tecnologie di livello enterprise della piattaforma Java. Ho affinato le mie capacità di organizzazione e programmazione. Ho potuto studiare a fondo le architetture software più utilizzate, nonchè tecnologie che durante il mio persorso universitario non ho potuto approfondire. Sono entrato in contatto con la realtà dello sviluppo di applicativi aziendali e del lavoro in team.

# Bibliografia


[Act15] Apache Software Foundation, sito ufficiale ActiveMQ - http://activemq.apache.org/

[Apa15] Apache Software Foundation, sito ufficiale - http://www.apache.org/

[Bug15] Bug tracking system, Wikipedia - http://en.wikipedia.org/wiki/Bug_tracking_system

[CDG06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber, BigTable : A Distributed Storage System for Structured Data - http://static.googleusercontent.com/media/research.google.com/it//archive/bigtable-osdi06.pdf

[Dyn15] DynamoDB, Wikipedia - http://en.wikipedia.org/wiki/Amazon_DynamoDB

[Eas15] EasyMarket travel solutions, sito ufficiale – http://www.easymarketcrs.it

[Etc13] European Travel Commission Digital Portal, Trends & Markets - http://etc-digital.org/digital-trends/ecommerce/travel-booking/regional-overview/europe/

[Fie00] Roy Fielding, Architectural Styles and the Design of Network-based Software Architectures - http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[GL02] Gilber S., Lynch N., Brewer's Conjecture and the Feasibility of Consistent Available Partition-Tolerant Web Services - http://doi.acm.org/10.1145/564585.564601

[Gla15] GlassFish, Wikipedia - http://it.wikipedia.org/wiki/GlassFish

[Gre13] Nicholas Greene, The five key advantages (and disadvantage) of NoSQL - http://greendatacenterconference.com/blog/the-five-key-advantages-and-disadvantages-of-nosql/

[Hyp15] Hypervisor, Wikipedia - http://it.wikipedia.org/wiki/Hypervisor

[J2e04] Java Community Process, Java 2 Platform Enterprise Edition - https://jcp.org/en/jsr/detail?id=151

[Jir15] Atlassian Jira, sito ufficiale - https://www.atlassian.com/software/jira

[Jme15] Apache Software Foundation, sito ufficiale Apache Jmeter http://jmeter.apache.org/

[Jms03] Java Community Process, Java Message Service API - https://www.jcp.org/en/jsr/detail?id=914

[Mav15] Apache Software Foundation, sito ufficiale Apache Maven - http://maven.apache.org/

[Mon15] MongoDB, documentazione online - http://docs.mongodb.org/manual/

[MyS15] MySQL, Funzionalità principali - http://dev.mysql.com/doc/refman/5.0/en/features.html

[MyW15] MySQL, pagina principale di MySQL WorkBench - http://www.mysql.it/products/workbench/

[Php15] PhpMyAdmin, sito ufficiale - http://www.phpmyadmin.net/home_page/index.php

[Str98] C. Strozzi, NoSQL, a Relational Database Management System - http://www.strozzi.it/cgi-bin/CSA/tw7/I/en_US/NoSQL/HomePage

[Sub15] Apache Software Foundation, sito ufficiale Apache Subversion - https://subversion.apache.org/

[Tom15] Tomcat, Wikipedia - http://it.wikipedia.org/wiki/Apache_Tomcat

[Vel15] Velocity, Wikipedia - http://it.wikipedia.org/wiki/Apache_Velocity

[Ver15] Controllo di versione, Wikipedia - http://en.wikipedia.org/wiki/Revision_control

[Vmw15] Vmware Inc. , vSphere ESXi - http://www.vmware.com/it/products/esxi-and-esx/overview.html