Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in
Ingegneria Informatica

Applicazione web per l'indagine sulla soddisfazione dei clienti

Customer satisfaction survey web application

Relatore: Prof. Giorgio Maria Di Nunzio

Laureando: Alex Tomasello

Anno Accademico 2012/2013

Indice

Capitolo 1

Introduzione

A partire dagli anni novanta del XX secolo, la tecnologia, lo sviluppo e la competizione nel mercato hanno portato le grandi aziende, soprattutto operanti nel settore terziaro del lavoro, ad avere la necessità di monitorare la qualità dei loro servizi.

La scelta di un approccio di mercato del tipo *customer oriented* rende necessaria l'effettuazione di indagini interne mirate a monitorare il modus operandi dei dipendenti, la competenza, il clima di gruppo, le risorse e le difficoltà. Di fondamentale importanza risultano anche le verifiche proposte per analizzare la soddisfazione e i bisogni dei clienti, al fine di proporre nuovi servizi o per migliorare quelli già offerti. In questo modo l'azienda può continuare ad essere non solo competitiva nel mercato, ma leader nel settore al quale appartiene. Permettere al cliente di esprimere un'opinione sulla qualità dei servizi offerti, dà la possibilità di mettere in evidenza quali siano le caratteristiche che fanno eccellere la ditta e favorisce l'individuazione degli aspetti su cui bisogna investire risorse per migliorarli. Al fine di cogliere e valutare il livello di soddisfazione è stato idealizzato il modello *SERVQUAL* da A. Zeithaml, A. Parasuraman, L. Berry.

Il *SERVQUAL* è un modello costituito da due serie di 22 item predefiniti con possibilità di risposta attraverso un valore numerico secondo una scala da 1 a 7. Le domande facilitano un confronto tra le aspettative generiche del cliente nei confronti del servizio e la percezione del prodotto offerto. Lo strumento consente di misurare il livello di soddisfazione su cinque elementi fondamentali del servizio: elementi tangibili (area di valutazione delle strutture fisiche, delle attrezzature e del personale); affidabilità (capacità di erogare il servizio promesso in modo affidabile e preciso); capacità di risposta (adeguatezza, volontà di aiutare i clienti e di fornire il servizio con prontezza); capacità di rassicurazione (competenza e cortesia degli impiegati e loro capacità di ispirare fiducia e sicurezza); empatia (assistenza premurosa e individualizzata che l'azienda riserva ai suoi clienti).

Per avere una valutazione quantitativa e qualitativa bisogna elaborare i dati e confrontare attese e percezioni.
I risultati del confronto fra attese (A) e percezioni (P) di qualità possono essere di tre tipi:

- P > A : la qualità del servizio è molto alta perchè le percezioni superano le

aspettative;

- P = A : la qualità del servizio è buona perchè si sono soddisfatte in pieno le attese del cliente;
- P < A : la qualità del servizio è bassa.

Lo strumento *SERVQUAL*, insieme ad altre metodologie centrate sul cliente, sulle sue aspettative e percezioni, hanno avuto ampia diffusione a livello internazionale. Lo sviluppo della tecnologia, e in particolar modo del settore informatico, hanno permesso in questi anni la rielaborazione di strumenti per l'analisi della soddisfazione dei clienti, quali quello sopra presentato, e la creazione di adattamenti o nuove applicazioni informatiche più rapide.

La mia tesi verterà proprio sulla presentazione di una nuova applicazione web realizzata per affiancare il *SERVQUAL* ed essere una via intuitiva ed immediata per dare all'Ente committente un feedback generale del suo servizio. L'applicazopne è stata sviluppata durante il periodo di stage che ho sostenuto presso SAIV S.p.A., in collaborazione con il tutor aziendale dott. Giovanni Lovato e Giulio Rigoni.

Nei prossimi capitoli presenterò il progetto nelle diverse fasi: analisi dei requisiti, scelte progettuali, descrizione del modello e strumentazione usata.

Capitolo 2

Descrizione del progetto

2.1 Analisi dei requisiti

L'applicazione è stata creata a seguito di una reale richiesta di una commessa da parte di un'azienda, che per diritti di privacy sostituirò con una banca fittizia, la Banca d'Europa, e ho adattato l'applicazione finalizzandola a monitorare i servizi offerti dalle varie sedi venete.
L'Ente utilizza già un questionario cartaceo aderente al modello *SERVQUAL*, ma richiedeva uno strumento intuitivo da affiancare al questionario stesso così da ottenere dei *feedback* anche dai clienti che ritenevano troppo impegnativa la compilazione del test. Un'applicazione quindi finalizzata a coinvolgere quanti più utenti possibili, ad avere una maggiore quantità di dati ed un'analisi immediata e automatizzata. Il nuovo modello di acquisizione dei dati non è stato specificato, e non ci erano stati imposti degli standard da seguire, validi per il monitoraggio del *customer satisfaction*. Al committente premeva presentare ai clienti uno strumento di rapido e facile utilizzo. La ditta è presente sul territorio nazionale con più sedi dislocate, ognuna delle quali dovrà essere fornita del nuovo strumento ad eccezione della sede centrale, l'unica autorizzata all'analisi e visualizzazione dei dati. Le informazioni raccolte verranno differenziate per filiale e periodo di osservazione. L'analisi dei voti dovrà essere consultata su grafici di diverso tipo. Inoltre l'applicazione da presentare ai clienti deve essere utilizzata su dispositivi touch screen o totem multimediali posti all'interno delle sedi. Infine richiedevano che fosse possibile inserire pubblicità a scopi commerciali, video ed informazioni di servizi e promozioni che l'azienda mette a disposizione.

2.2 Scelte progettuali

Dopo aver raccolto le richieste del committente, il team di sviluppo le ha analizzate per decidere come implementare l'applicazione ed offrire così una proposta di risoluzione al cliente. Sono stati considerati alcuni dettagli implementativi, i costi e il tempo di sviluppo necessario per la realizzazione.

Tenendo conto che l'applicazione doveva essere indipendente dalla piattaforma usata, in quanto doveva essere ospitata su dipositivi touch screen che possono essere sostituiti con prodotti concorrenziali o totem multimediali,la scelta è stata quella di sviluppare una *web application*, residente su un server, accessibile tramite browser web. I linguaggi, quindi, con cui abbiamo sviluppato il sistema sono HyperText Markup Language (HTML)[1], Cascading Style Sheets (CSS) [2] e JavaScript (JS) [3]. Il sistema di votazione è accessibile da tutte le sedi ed è composto da una sezione specifica per le offerte e pubblicità aziendali e da un'altra per l'inserimento del voto in modo anonimo. L'applicazione dedicata agli amministratori, distinta da quella di votazione, è finalizzata all'analisi dei dati e alla generazione dei grafici.

Essendo il sistema ancora in fase di sviluppo e dovendo ancora il committente analizzare e valutare la soluzione che abbiamo proposto, il modello dei dati salvati che useremo potrà essere cambiato e potrà, nei prossimi mesi, acquisire nuove proprietà. Per lo storage dei dati, dunque, avevamo il bisogno di usare un Database Management System (DBMS) non relazionale, orientato ai documenti e quindi non vincolato ad una struttura tabellare prefissata del modello rappresentato. L'utilizzo dei documenti per il salvataggio dei dati lascia maggiori libertà di modifica degli attributi del modello. Avendo deciso di sviluppare *web application*, la scelta del database per lo storage dei dati è ricaduta su CouchDB. Questo database è interrogabile tramite richieste HyperText Transfer Protocol (HTTP) e permette, inoltre, la sincronizzazione automatica tra database ospitati su host diversi. In questo modo è possibile sincronizzare i dati presenti nei database delle filiali con la sede centrale.

[1] http://www.w3.org/TR/REC-html40/
[2] http://www.w3.org/Style/CSS/
[3] http://www.w3.org/standards/webdesign/script

2.3 Presentazione del modello

La nostra applicazione è certamente più veloce ed intuitiva rispetto al modello *SERVQUAL*. Si tratta di un'applicazione che permette di avere un feedback generale e veloce, ma non completo. Per aderire al modello *SERVQUAL* sarebbe necessario elaborarlo ulteriormente, aggiungendo, per esempio, domande riguardanti le aspettative dei clienti rispetto i servizi offerti ed una scala più ampia come valore delle risposte. Altro punto di forza dell'applicazione è l'automatizzazione. A differenza del questionario cartaceo in cui poi risulta necessario registrare manualmente i dati, elaborarli ed archiviarli, l'applicazione permette queste operazioni in tempi veloci e contemporaneamente all'espressione del voto da parte del cliente, in modalità automatica. È necessario, però, evidenziare che il nostro modello non è esaustivo, ma è un'applicazione da affiancare ad altri strumenti.

L'utente ha la possibilità di esprimere la propria valutazione scegliendo tra le possibilità: non del tutto soddisfatto, mediamente soddisfatto, soddisfatto e molto soddisfatto. Ad ogni alternativa abbiamo conferito un valore: a "non del tutto soddisfatto" il valore 1, a "mediamente soddisfatto" il valore 3, a "soddisfatto" il valore 5 e a "molto soddisfatto" il valore 7. La scelta dei valori è stata fatta per permettere, in un tempo successivo, l'inserimento di altre valutazioni intermedie (2,4,6) che faciliterebbero una visione più completa e andrebbero a rispettare la scala di valutazione prevista da modello ServQual.

All'utente sarà permesso di esprimere il proprio voto in modo anonimo, intuitivo e veloce con la possibilità di aggiungere un commento che verrà registrato dal sistema. L'amministratore, invece, potrà analizzare i dati, anche in tempo reale, visualizzati su grafici.

Ad una prima analisi della commessa con il team, è stata presa la decisione di realizzare un'applicazione con una sezione *front end*, ottimizzata per essere eseguita su dispositivi touchscreen e una sezione *back end* accessibile solo dagli amministratori. Nel momento in cui ci siamo chiesti quali dati sarebbe stato utile salvare, abbiamo deciso di registrare nel sistema il commento, il nominativo e l'email dell'utente (dati presentati come facoltativi, con il modello per la privacy e l'utilizzazione dei dati), la data di votazione, la sede, il tipo di servizio e la valutazione espressa. Il commento personale permetterà all'ente di avere un riscontro più completo rispetto alla singola risposta di scelta multipla e molto utile al fine di una verifica dei servizi proposti, il nome e l'indirizzo di posta elettronica consentiranno al committente di contattare la persona che lo ha richiesto e la data di votazione può facilitare l'analisi dell'affluenza al servizio.

2.4 Classi

L'applicazione sviluppata è aderente al pattern architetturale Model View Controller (MVC), paradigma che prevede la distinzione di tre diversi componenti del sistema:

- **Model**: il modello è l'entità informativa del sistema e fornisce i metodi per accedere ai dati utili all'applicazione. Nel nostro caso il modello è composto dal voto del cliente, la data e la sede. I dati vengono registrati su database CouchDB.
- **View**: la vista presenta l'interfaccia grafica all'utente e ne determina l'interazione col sistema, visualizza i dati contenuti nel model e si occupa dell'interazione tra utenti e sistema. In questo caso è stata sviluppata in linguaggio HTML e CSS e permette di selezionare il livello di soddisfazione.
- **Controller**: il controllore gestisce la comunicazione tra l'interfaccia grafica e il database, in questo caso è stato sviluppato in linguaggio JS.

Il sistema di Customer Satisfaction (CS) proposto sarà composto da tre classi come descritto da Figura 2.1. Con *voting* si intende il processo di votazione , con *vote* il modello salvato su database.

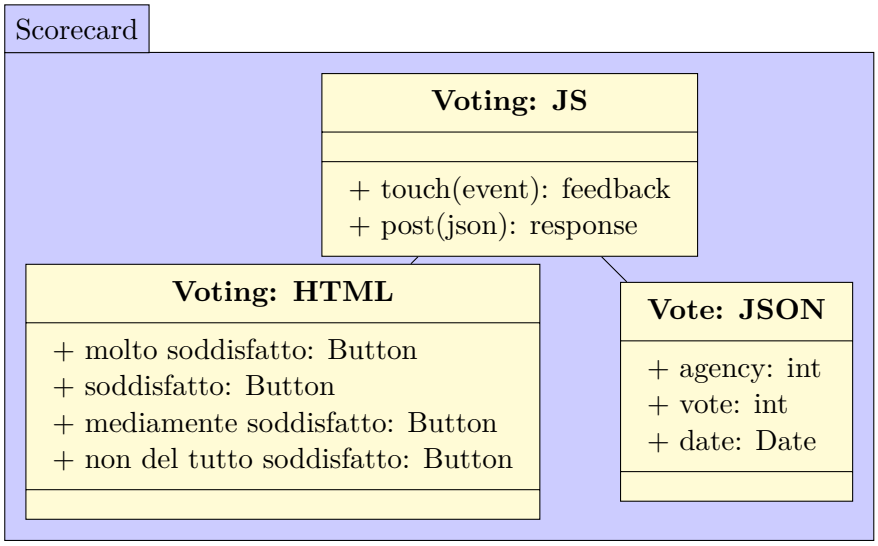


Figura 2.1: Classi del sistema di *CS*

L'utente agisce sulla vista del programma attivando una delle sue componenti di controllo, nel nostro caso i *bottoni* che permettono di registrare la votazione. Il controllore che è in ascolto dell'evento, riceve l'input di votazione ed esamina la vista per rilevarne le informazioni aggiuntive, come l'associazione tra pulsante premuto e valore assegnato. Il controllore invia tali informazioni al modello che effettua lo storage dei dati ed attende la risposta per aggiornare il proprio

stato. Successivamente richiede alla vista di visualizzare il risultato della computazione. La vista, infine, farà ritornare all'utente il risultato dell'operazione eseguita. L'implementazione del *controllore* in ascolto sull'evento votazione è riportato in figura 2.2, la sequenza del controllore del monitoraggio voti è riportato in figura 2.3.

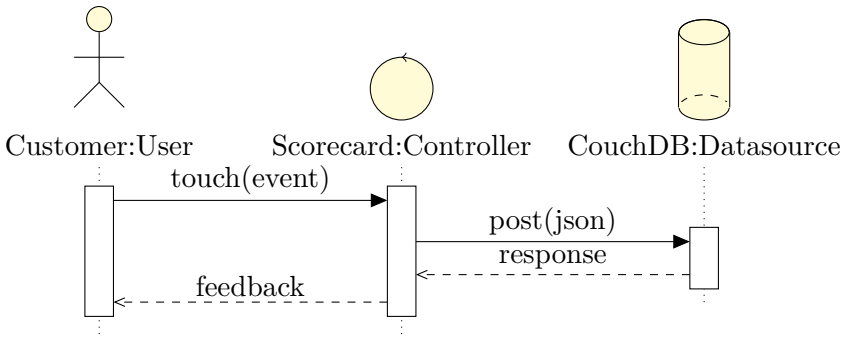


Figura 2.2: Sequenza di espressione voto

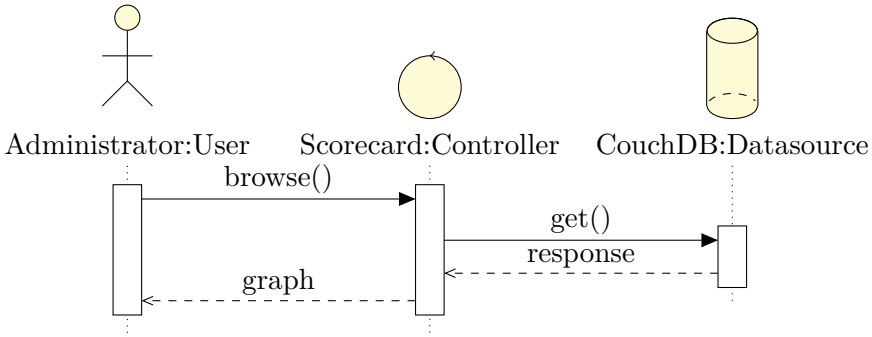


Figura 2.3: Sequenza di monitoraggio voti

2.5 Casi d'uso

Nell'applicazione qui presentata ci sono due casi d'uso: uno riservato agli utenti (Figura 2.4), l'altro agli amministratori (Figura 2.5).

2.5.1 Inserimento della votazione *front end*

- **Voto in uscita**: in uscita dalla sede il cliente troverà un totem multimediale touch screen con cui potrà esprimere il proprio livello di soddisfazione dopo aver usufruito dei servizi offerti. La schermata di introduzione è composta da un messaggio che spiega all'utilizzatore la finalità per la quale viene richiesto di lasciare un *feedback*, un video introduttivo realizzato dell'azienda a scopi commerciali e la scelta dell'operazione effettuata: servizio ricevuto presso lo sportello oppure una consulenza sulle operazioni finanziarie.

- **Inserimento commento**: l'utente può lasciare un proprio commento selezionando l'area con la voce "Lasciaci un commento". Comparirà una finestra di dialogo che permetterà l'inserimento del testo , il nome e cognome e l'indirizzo email nel caso in cui il cliente abbia piacere di essere contattato. Per registrare il commento bisogna accettare le condizioni sulla privacy.

- **Registrazione del voto**: il cliente esprime il proprio livello di soddisfazione tramite l'interfaccia selezionando uno dei 4 bottoni.

- **Feedback all'utente**: Terminato il processo di registrazione una finestra di dialogo certifica all'utente che il processo di voto si è concluso in modo corretto.

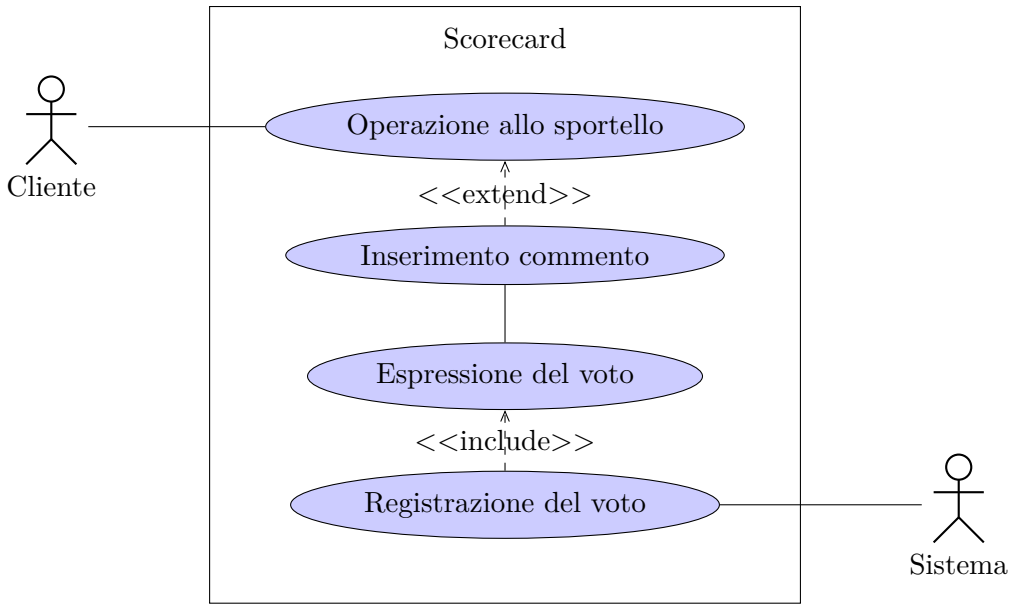


Figura 2.4: Caso d'uso votazione

2.5.2 Consultazione dei voti *back end*

L'amministratore potrà consultare l'andamento del livello di Customer Satisfaction (CS) via browser web. L'interfaccia di riepilogo dei voti presenterà in modo grafico i dati raccolti:

- l'amministratore accede all'interfaccia web del sistema.
- l'amministratore sceglie la/e sedi da monitorare, l'intervallo di tempo e la tipologia di grafico. Il grafico generato conterrà in ascissa l'intervallo di tempo scelto ed in ordinata il valore del CS. A seconda del periodo da monitare l'intevallo di campionamento è espresso in ore per la modalità "giornaliera", giorni per "mensile" e mesi per "annuale".
- la selezione del valore "automatico" aggiornerà il grafico ogni 5 secondi aggiornando i valori qualora venisse registrata una nuova votazione.

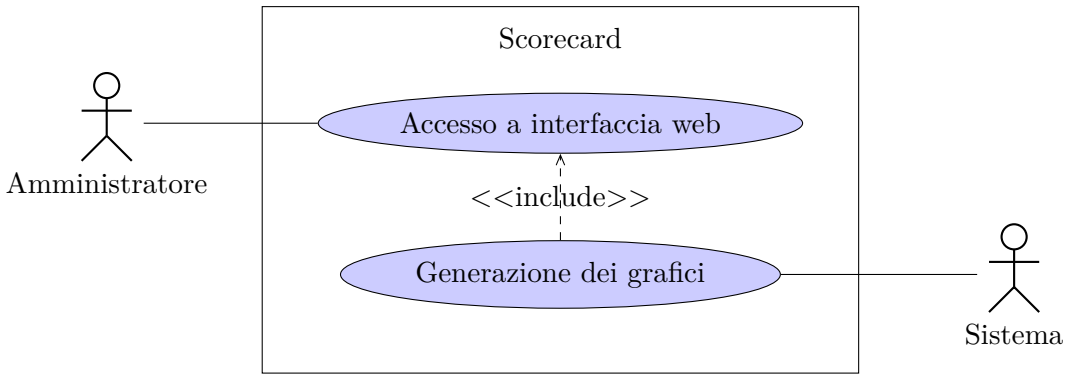


Figura 2.5: Caso d'uso consultazione dei voti

2.6 CouchDB

Apache Cluster Of Unreliable Commodity Hardware (CouchDB) [4] è un DBMS non relazionale ottimizzato per essere utilizzato in applicazioni web. È un database opensource, nato nel 2005 e mira a diventare il database di Internet.

CouchDB, essendo un database Not Only SQL (NoSQL), non memorizza i dati in tabelle ma ogni oggetto viene salvato su documenti utilizzando il formato JavaScript Object Notation (JSON). Ogni database è una collezione di documenti ed ogni documento contiene i propri dati e schema in modo indipendente dagli altri. Il documento memorizzato in una base di dati contiene tutte le proprietà dell'oggetto che si vuole rappresentare e, inoltre, memorizza i metadata associati ad esso : *"_id"* e *"_rev"*. La variabile *"_id"* rappresenta l'id univoco per ogni documento e, se non viene istanziata dall'utente, assume un valore assegnato dal motore di CouchDB. La proprietà *"_rev"* viene utilizzata per la gestione delle revisioni. CouchDB è in grado di gestire la contemporaneità di accesso ad una risorsa fra un elevato numero accssi in lettura e scrittura della risorsa stessa. Questo sistema è implementato tramite Multiversion Concurrency Control (MCC).

Proviamo ad immaginare il caso in cui un utente prova a leggere una risorsa e nello stesso istante un secondo utente la sta usando e modificando. Se questo passaggio critico non viene gestito in modo corretto, è possibile che il lettore veda la risorsa in modo inconsistente, in uno stato non completo e con una perdita parziale o totale dei dati. Questo tipo di situazione può, inoltre, creare stalli ed errori nell'applicazione che accede alla risorsa. Una soluzione per evitare ciò, è quella di bloccare la risorsa appena un utente prova accedere per modificarla. Limitare la risorsa ad un accesso contemporaneo non sempre è accettabile, in quanto blocca tutte operazioni di lettura e scrittura, rendendo inaccessibile la risorsa.

MCC propone un approccio differente: ogni utente che accede alla risorsa, ne visualizza il suo ultimo stato consistente, cioè uno stato in cui tutti i vincoli definiti dalla risorsa sono rispettati. In questo modo le modifiche effettuate da un utente non saranno visibili fino a che non viene completata l'operazione di scrittura. Tutti gli utenti, però, potranno accedere contemporaneamente alla risorsa. Per permettere ciò, ad ogni modifica non viene aggiornato direttamente il documento restituito dalla richiesta, ma lo si segna come obsoleto e se ne crea uno nuovo aggiornato, al quale verrà assegnato un nuovo valore della versione. La chiave *"_rev"* viene, dunque, aggiornata e questo meccanismo è alla base della prevenzione dei conflitti in fase di salvataggio. In questo modo ci sono più versioni dell'oggetto, ma solo una di esse è l'ultima versione. Per evitare sovraccarichi di documenti obsoleti è necessario che il sistema periodicamente faccia pulizia e cancelli i dati più vecchi. CouchDB predispone di un sistema di *merging* dei documenti per non perdere le modifiche fatte da diversi utenti alla stessa risorsa.

[4] http://couchdb.apache.org/

CouchDB fornisce la semantica *Atomicity Consistency Isolation Durability (ACID)* per tutte le transazioni. Una transazione è una sequenza di operazioni che può concludersi con un successo o un fallimento. In caso di successo il risultato delle operazioni deve essere permanente, mentre in caso di insuccesso si deve tornare allo stato della risorsa che aveva al momento dell'inizio della transazione. Aderire alla semantica ACID, permette di avere un sistema stabile e che mantiene il corretto funzionamento, anche a fronte di errori esterni, come ad esempio un mancanza della connessione durante l'invio dei dati, o crash del sistema. Di seguito l'analisi delle proprietà *ACID* :

- **Atomicity**: la transazione è indivisibile nella sua esecuzione e la sua esecuzione deve essere o totale o nulla, non sono ammesse esecuzioni parziali. In caso di fallimento dell'operazione si ritorna allo stato precedente;

- **Consistency**: la transazione porta l'oggetto sul quale si opera da uno stato consistente iniziale ad un secondo stato consistente finale. La transazione non deve violare eventuali vincoli di integrità, quindi non devono verificarsi contraddizioni tra i dati archiviati nel database;

- **Isolation**: ogni transazione deve essere eseguita in modo isolato e indipendente dalle altre transazioni. L'eventuale fallimento di una transazione non deve interferire con le altre transazioni in esecuzione;

- **Durability**: detta anche persistenza, garantisce che una transazione eseguita rimarrà valida anche in caso di errori successivi. I cambiamenti apportati non dovranno essere persi per crush o errori nel sistema. Per evitare che nel lasso di tempo fra il momento in cui la base di dati si prepara a scrivere le modifiche e quello in cui li scrive effettivamente si verifichino perdite di dati dovuti a malfunzionamenti, le operazioni eseguite verranno registrate su file di log.

2.6.1 JSON

I DBMS orientati ai documenti non memorizzano i dati in struttura tabellare con campi prefissati per ogni record come nei database relazionali, ma ogni valore è memorizzato come un documento con determinate caratteristiche. Qualsiasi numero di campi e qualsiasi tipologia di dato rappresentato senza essere definito a priori può essere aggiunto al documento.
JavaScript Object Notation (JSON) è un formato basato sul linguaggio *Javascript* utilizzato per lo scambio d'infomazioni in applicazioni client-server e utilizzato come formato da alcuni DBMS orientati ai documenti, quali CouchDB o MongoDB. Attraverso la notazione JSON è possibile rappresentare qualsiasi proprietà di un oggetto, con il vantaggio che la struttura del documento non è vincolata ad un modello prefissato, ma può variare nel tempo. Infatti è possibile aggiungere e rimuovere proprietà dell'oggetto o addirittura modificarne il tipo di dato rappresentato ed è possibile inserire attributi multipli o oggetti.

Un documento JavaScript Object Notation è formato da due strutture:

- Una collezione di coppie "nome"/"valore" ;
- Un elenco ordinato di valori.

Un oggetto è una collezone non ordinata di coppie "nome" : "valore". Un oggetto inizia con una parentesi graffa aperta "{" e finisce con una parentesi graffa chiusa '}". Un oggetto può essere asseganto come valore ad una variabile (Figura 2.6).

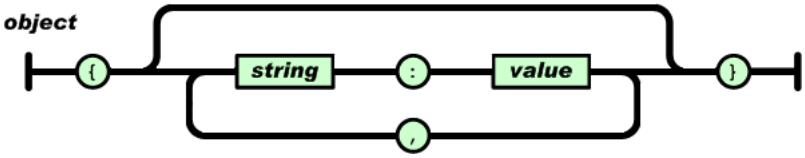


Figura 2.6: Oggetto JSON

Un array è un insieme ordinato di valori. Un array comincia con una parentesi quadra sinistra "[" e finisce con una parentesi destra"]". I valori sono separati da una virgola "," (Figura 2.7).

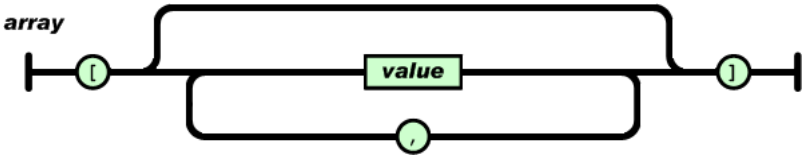


Figura 2.7: Array JSON

Ogni nome di variabile è seguito dai due punti ":" e le coppie "nome" : "valore" sono separati da una virgola "," (Figura 2.8). In CouchDB assumono significato particolare le variabili "_id" e "_rev".

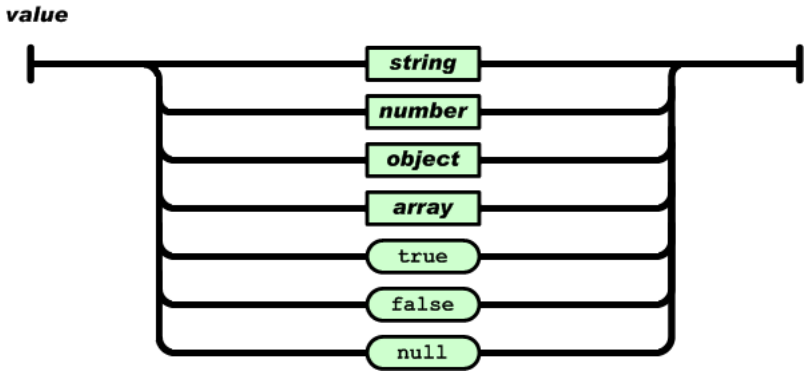


Figura 2.8: Valore di una variabile JSON

L'oggeto utilizzato nel nostro modello è composto da sei variabili:

- *agency*: indica la sede per la quale è stata registrata una votazione, il valore è una stringa univoca;
- *operation*: indica il tipo di servizio per il quale si vuole rilasciare un *feedback*;
- *score*: è una variabile numerica intera, nel nostro modello può assumere i valori {1,3,5,7};
- *date*: la data di effettuazione della votazione;
- *comment*: il commento rilasciato dall'utente, essendo un campo facoltativo, il valore di default sarà *null*;
- *customerName*: le credenziali del cliente che ha deciso di rilasciare un commento, anch'esso facoltativo;
- *customerEmail*: l'indirizzo mail del cliente, utilizzato dalla ditta committente per contattarlo in caso volesse approfondire l'indagine sulla soddisfazione del cliente.

Un esempio di oggetto salvato nel database:

```
1  {
2     _id:"182b00cd54a2b054fffabf1042000fbb",
3     _rev:"1-cb887f197900dce789c2dca898e4d330",
4     agency: "vicenza",
5     operation: "counter",
6     score: 5,
7     date: "2013-06-01T11:40:52.280Z",
8     comment: "Servizio efficiente",
9     customerName: "Alex Tomasello",
10    customerEmail: null
11 }
```

2.6.2 API

Il database è accessibile tramite richieste HyperText Transfer Protocol (HTTP), le cui proprietà sono specificate nella Request for Comments (RFC) 2616 [5]. Una richiesta HTTP è formata da una riga di richiesta (*request line*) che contiene il metodo, lo Uniform Resource Identifier (URI) e la versione del protocollo. La seconda componente della richiesta è la sezione *header* , la terza il *body*.

Attraverso i metodi messi a disposizione dal protollo, è possibile memorizzare nuovi dati e creare nuove viste (*Create*), richiedere informazioni (*Read*), modificare le informazioni memorizzate all'interno dei documenti (*Update*) ed eliminare documenti (*Delete*).

Ogni documento è raggiungibile da uno Uniform Resource Locator (URL) univoco. Il protocollo HTTP prevede la seguente sintassi per definire l' URL di una risorsa allocata :

http_URL = "http:" "//" host [":" port] [abs_path ["?" query]]

Host è l'indirizzo del server sul quale è ospitato il database, *port* indica la porta sulla quale è in ascolto CouchDB, che di default è la 5984 per richieste http, 6984 per ssl, modificabili accedendo al file di configurazione. Se si vuole accedere al database *abs_path* è il nome del database stesso, se si vuole accedere ad una vista il percorso da inserire sarà composto da

nome_database "/_design/" nome_design_document "/_view/" nome_vista.

Le operazioni Create Read Update Delete (CRUD) sulle risorse sono eseguibili tramite i seguenti metodi HTTP (gli esempi riportati sono stati realizzati con il command line tool *cURL*):

- **GET** : richiede l'elemento specificato. Il formato dell'URL definisce ciò che viene restituito: CouchDB può restituire elementi statici (ad esmepio viste), documenti di database e di configurazione. L'informazione viene restituita sotto forma di un documento JSON. Per ottenere documenti non l'URL deve essere composto da :

  "http://" host ":" port "/" database "/" _id

  es: *curl -X GET "http://atcustomersat:5984/padova/55090..."*.

  Per ottenere elementi statici come le viste l'URL deve essere composto da:

  "http://" host ":" port "/" database "/_design/"_id

  es: *curl -X GET "http://atcustomersat:5984/padova/_design/consulting"*.

- **HEAD** : questo metodo viene utilizzato per ottenere l'intestazione HTTP di una richiesta GET senza il corpo della risposta. Viene utilizzato per verificare l'esistenza di un oggetto senza richiederne il contenuto. In caso di documenti di grandi dimensioni, verificarne l'esistenza con il metodo HEAD velocizza notevolmente l'applicazione, in quanto non si deve aspettare che

[5] http://tools.ietf.org/html/rfc2616

ritorni l'oggetto richiesto. L'URL per accedere ai documenti ha la stessa sintassi di una richiesta GET.

es: *curl -X HEAD "http://atcustomersat:5984/padova/55090..."*

- **POST** : permette di inviare nuovi documenti al database di destinazione. In questo caso la richiesta POST deve avere come contenuto del body un documento JSON valido.

  Per inviare un documento l'URL deve essere composto da:

  "http://" host ":" port "/" database

  es : *curl -X POST "http://atcustomersat:5984/padova" -H "Content-Type: application/json" –data @scorecard.json*

  Dove *scorecard.json* è un file contentente un documento json.

- **PUT** : permette di inviare una risorsa specificata. In CouchDB il metodo PUT può essere utilizzato per creare nuovi dodumenti e database.

  es : *curl -X PUT "http://atcustomersat:5984/padova/123456798" -H "Content-Type:application/json" –data @scorecard.json*

- **DELETE** : cancella la risorsa specificata.

  es: *curl -X DELETE "http://atcustomersat:5984/padova/55090..."*

Per aderire alle specifiche HTTP, è necessario, inoltre, che gli headers siano settati e vengano forniti in modo da ottenere il formato e codifica corretta. Gli headers essenziali sono *Content - type* e *Accept*. **Content - type** specifica il tipo di contenuto dei dati che vengono forniti nella richiesta. CouchDB accetta il formato JSON quindi l'header sarà *application/json*. L'header **Accept**, invece, indica quali tipi di file sono accettati dal client, in formato MIME (es: text/html). Solitamente sarà settato con *application/json*, */* se sono supportati tutti i tipi di file.

Per realizzare interrogazioni parametrizzate, si usano le query-string. La query-string è la parte di un URL che contiene dei dati da passare in input ad un programma. Il carattere "?" apre la query-string, è utilizzato come carattere speciale e non viene incluso nella query. Ogni argomento è formato da una coppia chiave - valore. Alla chiave X si assegna il valore Y con il segno "=" ed ogni argomento è separato da un altro con il carattere "&". La sinstassi per realizzare una query string è, dunque, la seguente :
"?parametro1=valore1&parametro2=valore2&parametro3=valore3".

Le query parametrizzate vengono utilizzate nell'applicazione nel momento della generazione dei grafici. C'è la possibilità di scegliere la sede che si vuole monitorare, il tipo di servizio, l'intervallo temporale e il tipo di grafico. Ogni voce scelta andrà a settare un parametro nell'URL. I parametri permessi da CouchDB sono illustrati nella tabella seguente.

Parametro	**Valore**	**Default**	**Descrizione**
key	JSON	-	Indica la coppia chiave-valore, deve rispettare la sintassi dell'URL.
startkey	JSON	-	Indica il valore della chiave di partenza, deve rispettare la sintassi dell'URL.
startkey_docid	"_id" del documento	-	ID del documento di partenza.
endkey	JSON	-	Indica il valore massimo della chiave, deve rispettare la sintassi dell'URL.
endkey_docid	"_id" del documento	-	ID del documento che termina la ricerca.
limit	intero	-	Limita il numero di documenti restituiri.
descending	true/false	false	Ordinamento della ricerca.
skip	intero	0	Salta il numero indicato di documenti.
group	true/false	false	Controlla se la funzione di *reduce* viene applicata a un gruppo di chiavi distinte o singola riga di risultati.
group_level	intero	-	Indica il livello di raggruppamento da applicare alla funzione *reduce*.
reduce	true/false	true	Specifica se utilizzare la funzione *reduce*.

Ad esempio per ricevere i dati di una particolare sede, ragguppati per mesi e per il periodo temporale compreso tra gennaio e luglio 2013, la query è la seguente

?group_level=2&startkey=[2013,1,1,1]&endkey=[2013,7,31,23]

2.6.3 Replication

CouchDB consente di sincronizzare database situati su differenti host permettendo quindi agli utenti di accedere a risorse situate su host diversi in modo trasparente. La sincronizzazione è unilaterale, bisogna indicare un server che conterrà il database originale (*source*) e uno con funzione di *target*. Un database viene definito "locale" quando si trova sulla stessa istanza del server al quale viene inviata la richiesta di replication. Tutti gli altri casi i database sono definiti "remoti". Per sincronizzare un database si effettua una richiesta POST, inviando nel *body* un documento JSON contenente i database di source e destination:

```
//file replication.json:
{
  source:"database",
  target:"http://example.org/database"
}
```

curl -X POST http://atcustomersat:5984/_replicate -H "Content-Type: application/json" –data @replication.json

Quando si utilizza la sincronizzazione, CouchDB confronta i due database e verifica quali documenti presenti nel source differiscono da quelli presenti nel target. I documenti allocati nel target verranno modificati , cancellati o creati a seconda dello stato di quelli presenti nell'origine. Il numero di revisione del documento viene modificato, ed è possibile risalire alle modifiche effettuate al momento della sincronizzazione. Per realizzare una sincronizzazione bilaterale basta inviare due richieste di *replication* invertendo *target* e *source*.

Aggiungendo il parametro *continuous* e settandolo a *true*, è possibile continuare a sincronizzare i database ogni volta che vengono aggiunti o modificati documenti nel database di source. Nell'attuale versione di CouchDB (v1.3.0) non è possibile continuare la sincronizzazione dopo il riavvio del server, in quanto i settaggi della replication vengono resettati.

2.6.4 Map-Reduce

MapReduce è un framework software per scrivere facilmente applicazioni che elaborano grandi quantità di dati in parallelo. Il sistema è formato da due funzioni : *map* e *reduce*. La prima elabora le coppie chiave-valore , la seconda elabora il set di coppie generate dalla funzione *map* restituendo l'output richiesto.

- *map*: il nodo *master* prende i dati di ingresso (fase *input*), li suddivide in piccoli blocchi (fase *splitting*), generalmente da 64MB o 128MB e distribuisce il lavoro ai nodi *slaves*. Il singolo nodo *mapper* produce il risultato intermedio della funzione di map() sotto forma di coppie [chiave,valore] memorizzate su un file distribuito, la cui locazione è notificata al master alla fine della fase di map (fase *mapping*).

- *reduce*: il nodo *master* colleziona le risposte, combina le coppie [chiave,valore] in liste di valori che condividono la stessa chiave e li ordina per chiave (fase *shuffling*). Le coppie della forma, [chiave,IteratorList(valore,...)] sono passate ai nodi *reducer* che eseguono la funzione di reduce() (fase *reducing*).

Le funzioni di *map* e *reduce* sono implementate in Javascript, e CouchDB mette a disposizione ulteriori funzioni native da utilizzare. La funzione *reduce* come parametri di input ha una collection di chiavi, una collection di valori e un valore booleano *rereduce*. Se settato a true, rereduce permette di usare la funzione reduce al blocco di risultati generato dalle chiamate della funzione. Per database di grandi dimensioni, CouchDB chiama le funzioni per blocchi di dati in modo da velocizzare l'analisi dei dati. Non applicando la funzione rereduce c'è il rischio di ottenere un output sbagliato.

Ad esempio ipotizziamo di avere un set di documenti composto da 1000 chiavi. La fase di map calcola "chiave"="valore" e il motore di CouchDB divide i documenti ricevuti come input in 3 blocchi. Nella fase di reduce, la funzione calcola quanti elementi sono presenti nel db. La situazione di partenza prevede 1000 oggetti, la fase di map e reduce divide in 3 blocchi contenenti rispettivamente 333,333 e 334 oggetti. L'output resituito dalla funzione reduce è errato, in quanto calcola il numero di oggetti presenti restituito dopo le operazioni, calcola quindi 3 oggetti(Figura 2.9). Applicando la funzione di rereduce, invece, viene calcolata la somma dei valori restituiti dai 3 oggetti temporanei, restituendo l'output corretto(Figura 2.10).

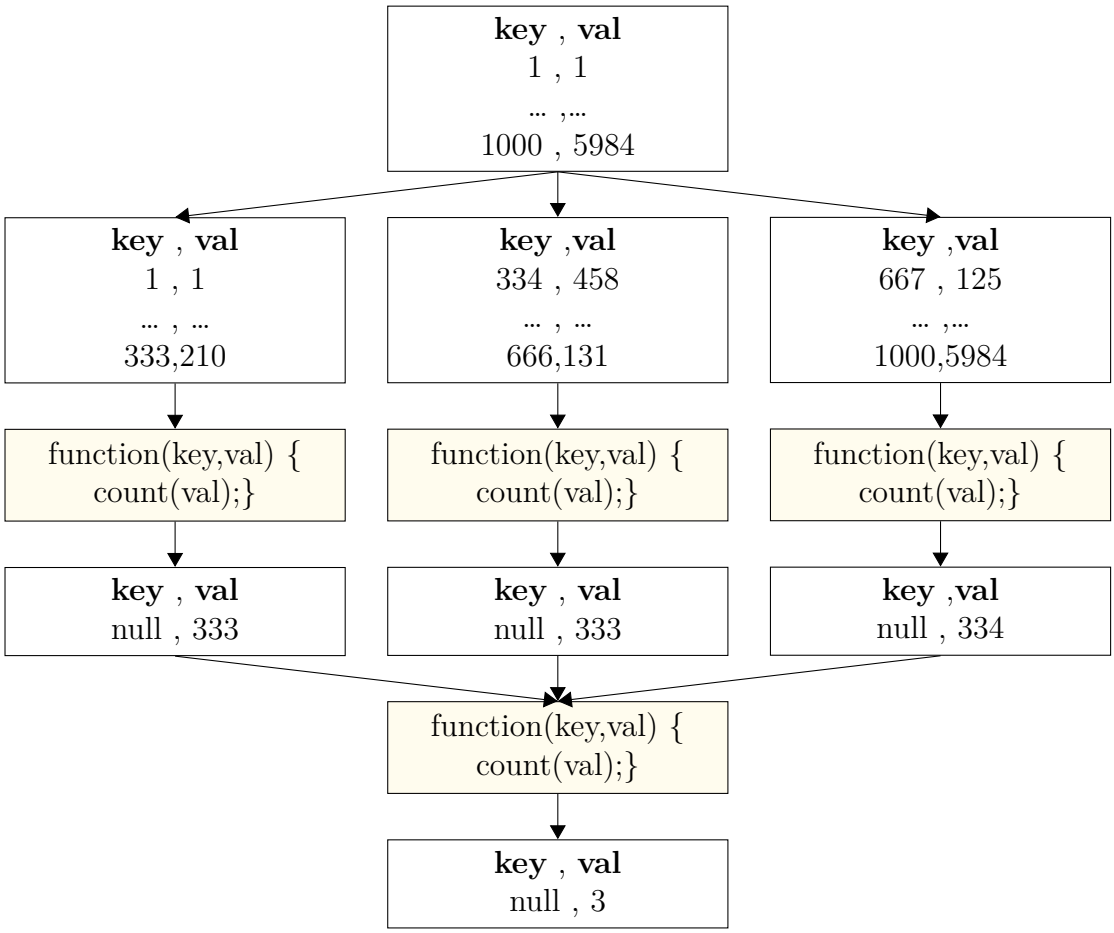


Figura 2.9: MapReduce con *rereduce=false*

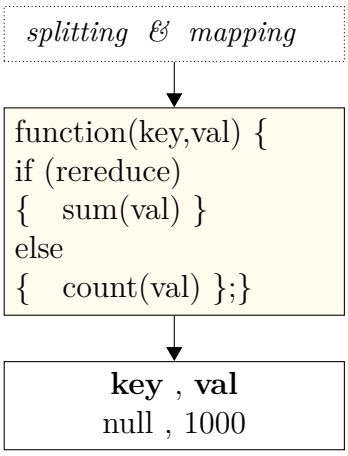


Figura 2.10: MapReduce con *rereduce=true*

Di seguito è presentata la funzione map utilizzata nel calcolo della somma dei voti per l'operazione di consulenza. Nel momento della fase di map, vengono scartati i documenti che non appartengono al gruppo scelto. Successivamente viene creata la coppia chiave-valore da inviare alla funzione di reduce. La chiave è la data trasformata in un array, per agevolare il raggruppamento in fase di output.

```
1  function(doc) {
2    if (doc.operation != 'consulting')
3      return;
4    var date = new Date(doc.date);
5    emit(
6      [date.getFullYear(),
7       (date.getMonth()  +1),
8       date.getDate(),
9       date.getHours()],
10     doc.score);
11 }
```

La funzione reduce raccoglie i dati, distinguendo i casi in cui rereduce è settato a true oppure a false.

```
1  function (key, values, rereduce) {
2    // Reduce function
3    var count = 0;
4    var sum = 0;
5    var i;
6
7    if(!rereduce) {
8      // `values` stores actual map output
9      for(i = 0; i < values.length; i++) {
10       count += 1;
11       sum += Number(values[i]);
12     }
13     return {"count":count, "sum":sum};
14   }
15
16   else {
17     // `values` stores count/sum objects returned previously.
18     for(i = 0; i < values.length; i++) {
19       count += values[i].count;
20       sum   += values[i].sum;
21     }
22     return {"count":count, "sum":sum};
23   }
24 }
```

2.7 Promise Pattern

Per registrare il voto espresso dall'utente si effettua una richiesta POST al server dove è ospitato il dabatase. Le richieste vengono inoltrate con metodologia Asynchronous JavaScript and XML (AJAX) [6]. AJAX permette di effettuare delle chiamate asincrone, cioè chiamate ad una risorsa esterna che non interferiscono con l'esecuzione della risorsa chiamante. I risultati della risorsa esterna saranno utilizzabili solo quando disponibili. Chiamate di tipo asincrono, quindi, permettono di effettuare altre operazioni senza alcun redirect o refresh della pagina.

La richiesta HTTP viene composta dal metodo $.ajax() prendendo in considerazione i parametri d'ingresso. Settando il metodo da chiamare, gli header necessari e il body, la richiesta viene composta e inviata al server. Utilizzare chiamate asicnrone, permette di strutturare il codice in nuovo modo: mentre le richieste AJAX vengono eseguite è possibile realizzare delle callback che possono essere chiamate a seconda dello stato della chiamata utilizzando il promise pattern.

Una promessa è un oggetto che rappresenta il valore di ritorno o l'eccezione lanciata che una funzione può eventualmente fornire. Il promise pattern prevede tre differenti stati di una promessa: soddisfatta (*fulfilled*) nel caso in cui la funzione ritorni un valore, non soddisfatta (*rejected*) nel caso la funzioni generi un'eccezione e in corso (*progress*). Lo stato progess è uno stato temporaneo, mentre fulfilled e rejected sono definitivi. Una promessa può passare da uno stato progess a uno degli altri due, ma non viceversa. Il costrutto delle promesse permette quindi di realizare delle funzioni che vengono chiamate a seconda dello stato dell'oggetto. Le funzioni sono create all'interno del metodo then(doneCallbacks [, failCallbacks] [, progressCallbacks]) riferito alla funzione iniziale.

Con l'uso delle promesse la scrittura del codice viene semplificata ed agevola il controllo degli errori. Se pensiamo ad una serie di eventi da realizzare in catena, una funzione viene scritta all'interno dell'altra, e richiede particolare attenzione dal parte del programmatore per evitare gli errori. Ad esempio il codice:

```
1 step1(function (value1) {
2     step2(value1, function(value2) {
3         step3(value2, function(value3) {
4             step4(value3, function(value4) {
5                 // Do something with value4
6             });
7         });
8     });
9 });
```

può essere semplificato con le promesse. Lo stato di una promessa viene propagato, quindi possiamo cogliere il fallimento di una di esse nel momento in cui

[6] http://api.jquery.com/jQuery.ajax/

fallisce, o alla fine della catena di promesse. In questo caso possiamo riscrivere il codice presentato prima con il seguente:

```
.then(step3)
.then(step4)
.then(function (value4) {
    // Do something with value4
}, function (error) {
    // Handle any error from step1 through step4
})
.done();
```

L'utilizzo delle promesse facilita anche l'utilizzo di chiamate parallele. Il metodo *all* riceve un array di promesse, ed appena tutte sono risolte si scatenano gli eventi della callback. In altri casi è possibile attendere che solo una di un array di promesse venga soddisfatta e si utilizza allora il metodo *any*. Un'ulteriore alternativa consiste nell'attendere un numero precisato di promesse soddisfatte utilizzando la funzione *some*.

Di seguito il codice utilizzato nell'applicazione per inviare la votazione. Il metodo ajax() riceve come parametri d'ingresso l'URL del server, il metodo HTTP e l'oggetto da memorizzare nella base di dati. In questo caso la promessa, quando e se viene soddisfatta, lancia la funzione che restituisce un feddback all'utente della corretta esecuzione del processo di voto.

```
$.ajax('http://' + server +':8888/' + agency, {
    type : "POST",
    contentType : "application/json",
    processData : false,
    data : JSON.stringify({
      "agency" : agency,
      "operation" : operation,
      "score" : $(this).data('score'),
      "date" : new Date(),
      "comment" : $('#comment').val(),
      "customerName" : $('#customerName').val(),
      "email" : $('#email').val()
    })
  }).then(function(response) {
      $('#feedback').modal({
        backdrop : 'static'
      });
    });
```

2.8 Screenshot

Il totem multimediale è posto in prossimità dell'uscita della filiale e la prima schermata che si trova di fronte l'utente è rappresentata in Figura 2.11. È composta da un messaggio introduttivo, un video di presentazione dei servizi offerti dalla ditta che viene riprodotto in automatico e ripetutamente e due pulsanti. L'utente per passare alla fase di rilascio del *feedback* dovrà selezionare uno dei bottoni, i quali indicano il tipo di servizio del quale ha usufruito. La distinzione fra servizio di sportello e consulenza è stata chiesta dal committente per monitorare le due differenti unità di lavoro presenti della filiale.

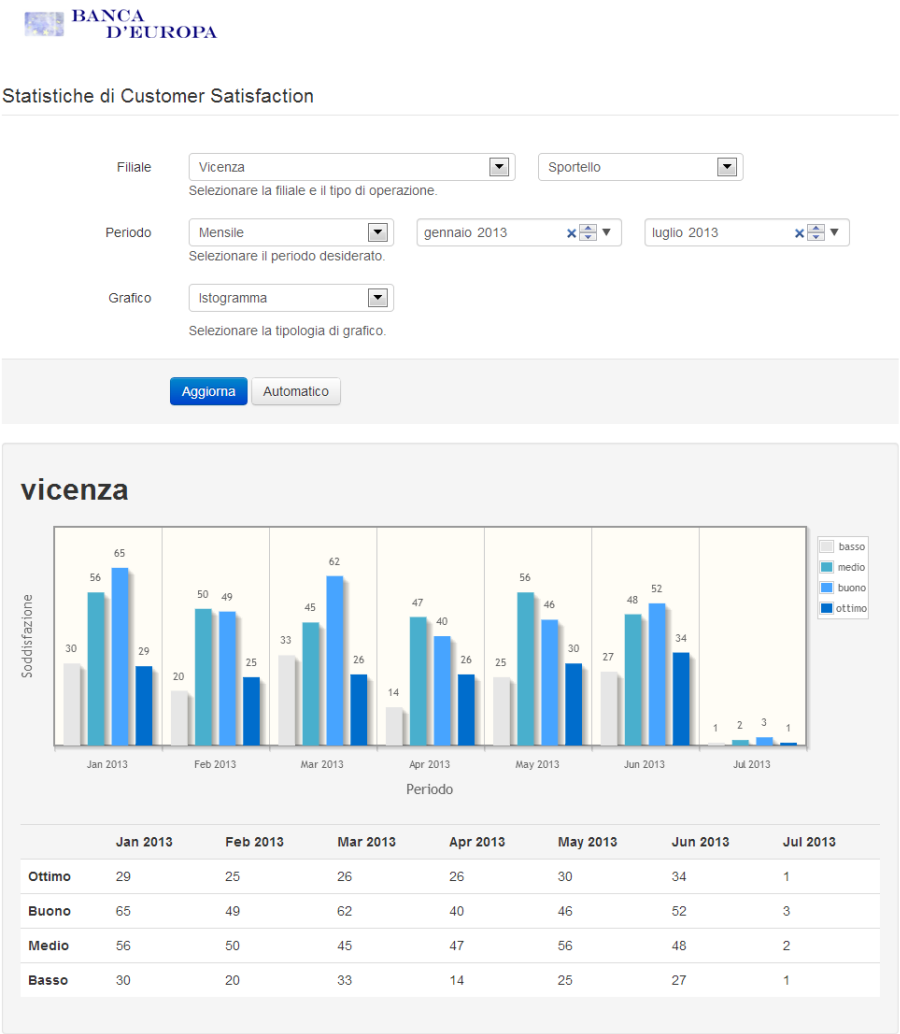
Figura 2.11: Schermata introduttiva

Una volta selezionato il tipo di operazione, all'utente verrà data la possibilità di rilasciare un commento e la votazione, Figura 2.12. Per rilasciare il commento, l'utente deve selezionare la *textarea* che riporta la voce "Tocca qui se vuoi lasciare un commento", comparirà una nuova schermata dove potrà inserire il commento, i propri dati e l'indirizzo mail. Per inviare il commento deve accettare il modulo per la privacy proposto della ditta Figura 2.13.

Figura 2.12: Scelta del livello di soddisfazione

Figura 2.13: Inserimento del commento

L'amministratore accede alla pagina dedicata alla raccolta dei dati e creazione dei grafici. Avrà la possibilità di scegliere la filiale da monitorare e il gruppo di lavoro, sportello o consulenza. Sono tre le tipologie di grafico che possono essere visualizzate : istogramma, con in ordinata il numero totale di voti espressi raggruppati per valore, grafico con andamento lineare, per monitorare la media, e grafico a torta con il totare di voti espressi. In ascissa, l'intervallo temporale cambia a seconda del periodo di osservazione scelto dall'amministratore: è possibile monitorare le singole ore della giornata oppure un intervallo di giorni, settimane o mesi.



	Jan 2013	Feb 2013	Mar 2013	Apr 2013	May 2013	Jun 2013	Jul 2013
Ottimo	29	25	26	26	30	34	1
Buono	65	49	62	40	46	52	3
Medio	56	50	45	47	56	48	2
Basso	30	20	33	14	25	27	1

Figura 2.14: Generazione dei grafici

Capitolo 3

Conclusioni

Le modalità di sviluppo e la scelta degli strumenti hanno premesso di creare un'applicazione modulare. Modelli e procedure sono state più volte modificate nel corso della progettazione e implementazione. La scelta di usare un database non relazionale, di aderire al pattern architetturale Model View Controller (MVC) e al *Promise Pattern*, ha permesso di elaborare un'applicazione solida e facilmente modificabile.

Il modello stesso dei dati salvato nel database CouchDB è stato più volte cambiato, aggiungendo e modificando proprietà dell'oggetto senza riscontrare difficoltà nella definizione dei tipi di dati. Diversamente sarebbe stato se avessimo utilizzato un database relazionale. CouchDB si è dimostrato una scelta valida anche per quanto riguarda la sincronizzazione dei dati. Trovandosi di fronte a una situzione in cui più host dovevano dialogare fra di loro, i metodi, facilmente configurabili, messi a disposizione dal database hanno agevolato il passaggio di dati fra sede centrale e filiali.

La struttura aderente al pattern MVC a promise pattern, invece, permette di inserire nuove azioni che vanno a dialogare con i modelli. Ogni funzione viene implementata in modo indipendente dalle altre, in quanto lo stato di una promessa può essere fulfilled, restituendo il contenuto, oppure rejected. Funzioni concatenate fra loro devono solo gestire i due casi appena citati. Saranno implementate nuove funzioni per gestire il login per gli amministratori al momento della visualizzazione dei grafici. Questa sezione verrà implementata utilizzando un proprio framework PHP: Hypertext Preprocessor (PHP) in fase di sviluppo che gestirà la fase di autenticazione, implementazione di Access Control List (ACL) e la sicurezza dei dati. Ulteriori funzionalità verranno aggiunta per quando riguarda l'analisi dei dati, permettendo di confrontare fra loro più sedi. L'utilizzo di richieste Asynchronous JavaScript and XML (AJAX) parallele consente già di ottenere tutti i dati necessari per la composizione di nuovi grafici.

L'applicazione ha soddisfatto le attese del cliente committente e allo stato attuale è installata in versione di prova in alcune sedi. Si è cercato inoltre di implementare il codice in modo da lasciare ampio spazio alle customizzazioni, così da ottenere un modello valido anche per altre utilizzazioni future.

Capitolo 4

Bibliografia e sitografia

- Zeithaml Parasuraman Berry, *Delivering Quality Service: Balancing Customer Perceptions and Expectations*, Free Press, 2009
- Busacca B., *Le risorse di fiducia dell'impresa: soddisfazione del cliente, creazione del valore strategie di accrescimento*, Utet, 1994
- http://wiki.apache.org/couchdb/Reference
- http://couchdb.readthedocs.org/en/latest/
- https://github.com/kriskowal/q/
- http://www.ensta-paristech.fr/ kielbasi/tikzuml/index.php?lang=en
- http://en.wikipedia.org/wiki/Unified_Modeling_Language
- http://research.google.com/archive/mapreduce.html
- http://api.jquery.com/jQuery.ajax/
- http://tools.ietf.org/html/rfc2616
- http://www.json.org/
- http://en.wikipedia.org/wiki/Model-View-Controller

Capitolo 5

Acronimi

Customer Satisfaction (CS)
HyperText Markup Language (HTML)
Cascading Style Sheets (CSS)
JavaScript (JS)
Database Management System (DBMS)
HyperText Transfer Protocol (HTTP)
Model View Controller (MVC)
Cluster Of Unreliable Commodity Hardware (CouchDB)
Not Only SQL (NoSQL)
JavaScript Object Notation (JSON)
Multiversion Concurrency Control (MCC)
Atomicity Consistency Isolation Durability (ACID)
Application Programming Interface (API)
Request for Comments (RFC)
Uniform Resource Identifier (URI)
Uniform Resource Locator (URL)
Create Read Update Delete (CRUD)
Asynchronous JavaScript and XML (AJAX)
Access Control List (ACL)
PHP: Hypertext Preprocessor (PHP)