Università degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria "Enzo Ferrari" Modena

Corso di Laurea Magistrale in Ingegneria Informatica

Progetto e realizzazione della Visual Analytics Dashboard orientata a Web Analytics e Search Engine Optimization

Relatore:
Prof. Sonia Bergamaschi

Correlatore:
Ing. Alessandro Pelliciari

Tesi di Laurea di:
Matteo Senardi

Anno Accademico 2015/2016

“Un bug non è mai solo un errore. È qualcosa di più …
Un pensiero sbagliato che ti rende quello che sei.”

Indice

RINGRAZIAMENTI

Nello scrivere i ringraziamenti si stanno accavallando nella mia mente una serie di momenti indimenticabili di questo percorso durato 2 dei 5 anni nella facoltà di Ingegneria. Attimi di tensione congiunti a risate prima e dopo un esame, anticipati da lunghe e allo stesso tempo serrate preparazioni tra libri, appunti, sorrisi, studio e pasti di gruppo. Dal mattino alla sera dei giorni feriali, dei fine settimana o in vacanza, non importava. Ricordi di sacrifici e soddisfazioni.

Tutto questo sarebbe stato impossibile senza il supporto di alcune persone che desidero ringraziare. In primis ringrazio la Professoressa Sonia Bergamaschi che mi ha concesso di approcciarmi e di approfondire l'argomento del Web Marketing con grande interesse, seguendo interamente l'intero lavoro inerente il progetto svolto durante il tirocinio.

Voglio inoltre fortemente ringraziare tutta la mia famiglia, ed in particolare mia mamma che non mi ha mai fatto mancare nulla di cui avessi bisogno durante tutta la mia vita e di conseguenza nel mio intero percorso di studio, rivelandosi un fondamentale e amabile sostegno. Senza la mia famiglia, che mi ha supportato sia dal punto di vista economico che psicologico, oggi non potrei festeggiare questo importante traguardo.

Un grazie speciale va a Greta, la mia ragazza, punto fermo costante da ancor prima che iniziassi questo percorso. Meglio di chiunque altro, mi ha spronato e reso più leggero durante ogni difficoltà. Mi ha fatto riflettere, crescere, e tornare bambino, non smetterò mai di emozionarmi per lei.

Grazie ai miei amici di sempre per i bellissimi momenti passati insieme. Grazie a tutti i compagni di Università, in particolare Lorenzo, Giovanni e Gianluca, per aver reso stimolanti e divertenti questi anni di corso.

Un sincero ringraziamento va a Webranking e a tutti i colleghi. In particolare nella persona di Alessandro, mentore e amico, che ha mi permesso di svolgere con entusiasmo l'intero percorso di tirocinio. Grazie inoltre ai miei colleghi del reparto Innovation, Luca e Alberto, per aver creato insieme ad Alessandro un gruppo di lavoro coeso.

ABSTRACT

Obiettivo della tesi è stato la progettazione e la realizzazione di una Web application denominata Visual Analytics Dashboard, nell'ambito dello stage svolto presso Netidea Webranking s.r.l.

L'azienda è tra le realtà più conosciute e apprezzate in Italia per i servizi di Search Marketing di alto profilo, ed è anche l'agenzia italiana più certificata da Google per i servizi di Web Marketing in un gruppo selezionato di 50 top agenzie nel mondo.

La Visual Analytics Dashboard appositamente creata predispone di un accesso multi-utente e multi-cliente, al fine di raccogliere e rappresentare dati provenienti da fonti eterogenee.

Nello specifico si tratta di dati relativi alle interazioni degli utenti su siti Web di clienti committenti, denominato *Traffico Web Analytics*, e ai posizionamenti inerenti la ricerca di parole chiave d'interesse su motori di ricerca, *SEO* (*Search Engine Optimization*).

L'elemento imprescindibile ai fini della realizzazione dell'applicazione è stato il monitoraggio e l'aggregazione di un elevato numero di dati, relativi a circa 250 siti Web dei vari clienti.

La complessità derivante dalla dimensione del *dataset* ha richiesto un approccio efficace e innovativo, volto alla gestione e alla presentazione di indicatori chiave di performance (*Kpi*), stabiliti sulle specifiche e peculiari esigenze di ciascun cliente.

In aggiunta è stato creato un sistema di confronto dei dati su archi temporali di lunghezza identica e non, e su gruppi di siti appartenenti a paesi e/o brand distinti.

L'implementazione della Visual Analytics Dashboard ha preso forma seguendo un approccio di sviluppo *Agile*, volutamente guidato da test (*Test Driven Development*) e di tipo iterativo *simil-Scrum*, fondato quindi sul raggiungimento di obiettivi comuni passo dopo passo.

In seguito al rilascio dell'applicazione, il monitoraggio degli errori in tempo reale, tramite sistema di terze parti, ha contribuito alla risoluzione dei bachi (*bug*) riscontrati durante le interazioni di utenti/clienti.

Il continuo processo di ottimizzazione ha concorso al successo del lavoro svolto e alla soddisfazione dei clienti.

Come nota finale, l'approccio seguito è a nostra conoscenza totalmente innovativo in termini di funzionalità.

1 INTRODUZIONE

Data la continua diffusione di Internet, cui segue un'espansione costantemente crescente degli strumenti atti alla raccolta di statistiche per la descrizione della qualità dei brand online, la quantità di dati prodotta quotidianamente da ciascun dispositivo connesso in rete risulta in perenne aumento.

Qualsiasi tipo di azione, da un semplice click sui motori di ricerca alla navigazione nei siti, può comportare infatti una procedura di raccolta e analisi.

Un sito Web è una tipologia di prodotto che offre un vantaggio rispetto a tutti gli altri prodotti, non solo software.

Mentre con la maggior parte dei prodotti è possibile avere delle statistiche di funzionamento e di gradimento del prodotto stesso solitamente molto vaghe (si pensi ad esempio alle inserzioni sui quotidiani), un sito Web permette per contro di ottenere statistiche estremamente precise, specie al rilascio, momento che per un sito Web coincide con la sua pubblicazione.

Il procedimento di analisi del dato che porta al processo di ottimizzazione di un sito web è stato studiato nel corso del tirocinio svolto presso Netidea Webranking s.r.l. di Correggio (RE).

L'azienda che vanta già 18 anni di esperienza è leader in Italia per i servizi di Search Marketing di alto profilo, ed è anche l'agenzia italiana più certificata da Google per i servizi di Web Marketing, in un gruppo selezionato di 50 top agenzie nel mondo.

Il processo di ottimizzazione di un sito nasce a seguito dell'introduzione di due nuove discipline in ambito Web: la *Web Analytics* (*WA*) e la *Search Engine Optimization* (*SEO*).

Web Analytics è la scienza fondata sull'utilizzo e l'analisi di dati per individuare trend passati e futuri, al fine di migliorare le performance presenti sulla base di sistemi di tracciamento che effettuano l'analisi degli utenti di un sito Web.

Scopo primario è quindi la profilazione dell'utente per finalità statistiche o di marketing mirato, ne segue che la comprensione dell'approccio orientato su sistemi di *Web Analytics* consente di acquisire vantaggiose informazioni per cogliere i dati delle visite.

Con il termine *Search Engine Optimization* si intendono tutte quelle attività finalizzate ad ottenere una migliore accessibilità, architettura e comprensione del sito Web da parte dei motori di ricerca.

Si tratta quindi un metodo per strutturare siti e contenuti in modo da incrementare la rilevanza e la visibilità del sito d'interesse, garantendone un posizionamento migliore e quindi una maggiore esposizione nei motori di ricerca.

La procedura *SEO* si riferisce principalmente ai cosiddetti posizionamenti organici (dall'inglese *organic placement*) che mediante azioni di ottimizzazione del sito per i motori di ricerca offre, se ben eseguita, un ottimo ritorno di investimento.

Il primo passaggio, logico ma anche temporale, del lavoro svolto durante il tirocinio ha riguardato la definizione del caso di studio da prendere in esame.

Obiettivo della tesi è stato la progettazione e la realizzazione di una Web application denominata Visual Analytics Dashboard, nell'ambito della raccolta dei dati indicatori volti al miglioramento delle performance di un sito.

Due sono i clienti che hanno sottoposto il problema a Webranking, operanti in campo farmaceutico e automotive, che per ragioni di privacy chiamerò rispettivamente Acme e Oceanic Airlines. Si tratta di compagnie internazionali altamente articolate, con una complessità tale da influire indirettamente, ma anche consequenzialmente, le aree di *Web Analytics* e *SEO.*

Questa forte motivazione ha quindi portato allo sviluppo della dashboard in questione.

Per definizione una dashboard è uno strumento interattivo di raccolta, monitoraggio e visualizzazione di dati ed informazioni.

L'obiettivo primario della applicazione progettata si focalizza sull'aggregazione e la visualizzazione dei dati appartenenti a più di 250 siti diversi dei due committenti.

La difficoltà che scaturisce dalla dimensione del *dataset* ha necessitato di un metodo nuovo, incisivo e funzionale finalizzato alla gestione e alla presentazione di indicatori chiave di performance (*Kpi*), stabiliti sulle esigenze puntuali di ogni cliente.

I dati raccolti eterogenei per natura, contemplando sia indicatori *Web Analytics* da un lato che posizionamenti di siti Web dall'altro, hanno fortemente guidato l'implementazione dell'applicazione.

Tale eterogeneità dei dati si è conseguentemente riflessa sulla modalità di raccolta degli stessi, rendendosi necessaria l'integrazione informazioni relative a API diverse e account diversi, in considerazione all'alto numero di siti coinvolti su paesi e brand distinti.

In tale ottica sono state valutate anche le soluzioni attualmente presenti in commercio, ma a nostro avviso le richieste avrebbero dovuto imprescindibilmente prevedere un approccio del tutto innovativo in termini di funzionalità.

Al fine di raccogliere e rappresentare dati provenienti da fonti eterogenee, la Visual Analytics Dashboard appositamente creata predispone di un accesso multi-utente e multi-cliente.

Ai requisiti essenziali di progettazione, vanno ad aggiungersi diverse azioni di ottimizzazione volte al mantenimento e alla pulizia del dato scaricato esente da campionamento.

È stato inoltre creato un sistema atto a consentire un duplice confronto: da un lato la comparazione tra dati aventi archi temporali di lunghezza identica e non, dall'altro il raffronto su gruppi di siti appartenenti a paesi e/o brand distinti.

Sulla base di questi requisiti si è quindi giunti alla fase di progettazione e realizzazione della soluzione individuata.

Per quanto riguarda le scelte tecnologiche per la messa a punto della Visual Analytics Dashboard si è fatto riferimento a diversi strumenti, implementati ad hoc per consentire l'aggregazione e la presentazione dei dati memorizzati.

Definita Amazon come infrastruttura di servizio ospitante l'applicazione Web, si è passati alla definizione di tre livelli logici, sviluppati e mantenuti come moduli indipendenti.

Lato logica dei dati, si è fatto uso congiunto di MySQL sistema di database relazionale, e di MongoDB sistema di database NoSQL orientato alla memorizzazione dei documenti. Sottolineo che questo approccio è stato adottato previa valutazione di due soluzioni alternative, la cui analisi troverà spazio in seguito.

Lato logica applicativa, si è ricorsi all'adozione del linguaggio di programmazione Python in combinazione con il framework Web-based Django. Si cita inoltre il supporto di librerie aggiuntive create per il calcolo scientifico, utilizzate principalmente per le operazioni di aggregazione e filtro sui dati.

Lato presentazione finalizzato alla realizzazione dell'interfaccia utente, con particolare riferimento alla presentazione delle statistiche, si è fatto uso di diverse librerie JavaScript. Su tutte, il framework AngularJS volto alla semplificazione del caricamento dinamico delle risorse in pagina quando necessario, solitamente in risposta ad azioni dell'utente.

Come nota finale necessaria al rilascio dell'applicazione in produzione si è valutato la piattaforma software per lo sviluppo di applicazioni Web.

Dato poi che funzionalità e prestazioni dell'applicazione sviluppata si sono rivelate tutt'altro che trascurabili, punto cruciale della fase di sviluppo è stato la realizzazione di test finalizzati alla valutazione dei molteplici approcci che si sarebbero potuti utilizzare.

Lo sviluppo dei test ha preso forma a partire, principalmente, dalla teoria denominata *Test Automation Pyramid* di Mike Cohn che sarà esposta successivamente.

L'implementazione della Visual Analytics Dashboard è stata attuata un approccio di sviluppo *Agile*, volutamente guidato da test (*Test Driven Development*) e di tipo iterativo *simil-Scrum*, basato pertanto sulla concretizzazione di obiettivi comuni step by step.

Di fondamentale importanza sono stati poi i test volti allo stress della macchina ospitante l'applicazione, nonché al miglioramento delle relative performance.

In seguito al rilascio dell'applicazione si è quindi rivolta l'attenzione al rilevamento di bachi (*bug*), riscontrato durante le interazioni degli utenti/clienti.
Il monitoraggio degli errori in tempo reale, tramite sistema di terze parti Sentry, ha concorso attivamente alla ricezione di questi feedback.
Il continuo processo di ottimizzazione ha contribuito al successo del lavoro svolto, con grande apprezzamento da parte dei clienti.

Per semplicità di lettura, a conclusione di questa parte introduttiva riporto un breve accenno ai temi trattati nelle varie sezioni.
Nel Capitolo 2 è definito il background degli argomenti di analisi, con particolare riguardo ai sistemi di Web Marketing studiati.
Nel Capitolo 3 è presentata l'analisi del problema relativo al caso di studio, ponendo accento allo sviluppo dell'applicazione innescato proprio dalle soluzioni individuate.
Nel Capitolo 4 sono mostrati progetto ed implementazione della Visual Analytics Dashboard.
Nel Capitolo 5 è evidenziato l'approccio di sviluppo continuo fortemente guidato da test, volto a migliorare funzionalità, prestazioni, esperienza utente e alla risoluzione degli errori riscontrati dopo il rilascio.
Nel Capitolo 7 a conclusione dell'attività svolta, desidero fare rilevare i risultati ottenuti nonché i possibili scenari e sviluppi futuri nell'incrementale processo di evoluzione della Visual Analytics Dashboard, d'ora in poi denominata *VAD*.

2 BACKGROUND

La sezione presenta un excursus agli argomenti affrontati durante lo sviluppo della Visual Analytics Dashboard (*VAD*).

2.1 INTRODUZIONE A WEB ANALYTICS E SEO

La *Web Analytics* è la scienza fondata sull'utilizzo e l'analisi di dati per individuare trend passati, migliorare le performance presenti e fare previsioni sugli andamenti futuri. In particolare si basa su sistemi di tracciamento che effettuano un'analisi dei fruitori di un sito Web volta alla profilazione dell'utente per finalità statistiche o per effettuare marketing mirato.

Non conta quindi lo scopo intrinseco del sito Web, poiché tramite la *Web Analytics* è possibile comprendere la cosiddetta *customer journey*, intesa come rotta degli utenti ad ogni interazione.

La comprensione dell'approccio dei sistemi di *Analytics*, può fornire informazioni importanti per cogliere le performance delle visite. Per esempio, in caso di forte declino potrebbe essere possibile identificare il problema (mettiamo imputabile alla presenza di un competitor) e cercare soluzioni sfruttando la pubblicità a pagamento o migliorando il posizionamento sui motori di ricerca, di cui si parlerà in seguito.

Alcuni dei dati raccolti provengono dal sito stesso, come l'URL delle pagine visitate dall'utente. Altri dati sono recepiti a partire dalle informazioni presenti nel browser dell'utente, quali lingua, tipo di browser, dispositivo e sistema operativo usato per accedere al sito. In questo modo è possibile ricostruire la statistica della percentuale di utenti che utilizzano Mac o Pc, iPhone o Android.

In aggiunta possono essere memorizzate informazioni circa il contenuto visitato e il tempo di navigazione, oltre all'individuazione della sorgente che ha portato l'utente sul sito.

I passi chiave per ottenere analisi mirate sulle utenze devono prevedere (Fig. 1):

1. identificazione degli obiettivi del tracciamento (fidelizzazione, vendita, ricerca di nuovi utenti …);
2. raccolta dei dati;
3. analisi delle performance, in particolare se e come filtrare o confrontare i dati raccolti (click di iscrizione a newsletter, confronto con mese precedente);
4. decisone sulle scelte strategiche future.

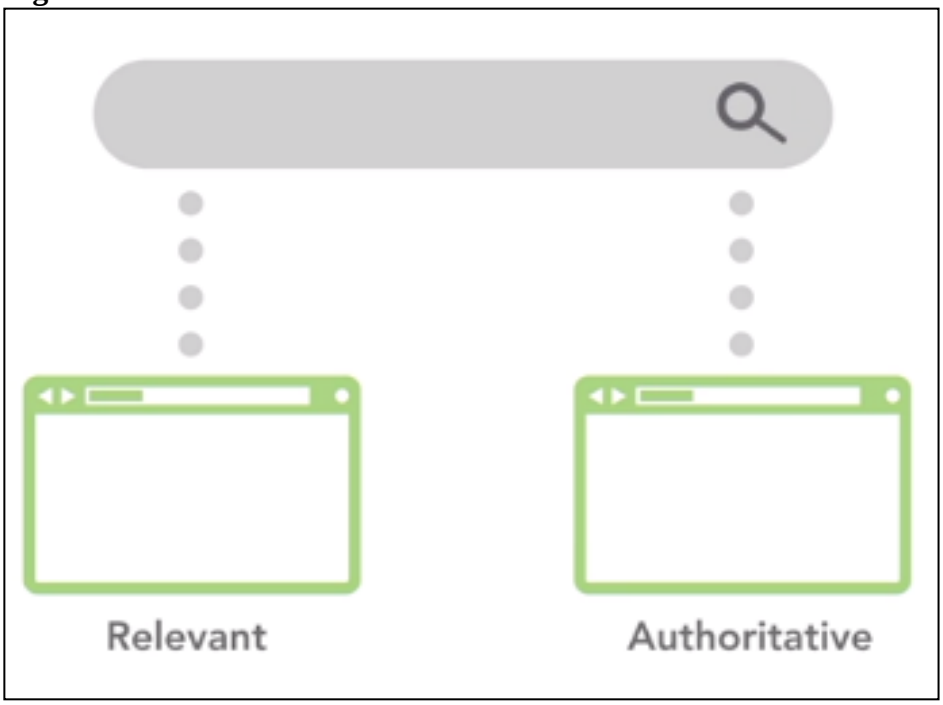
Figura 1:

Con il termine *SEO* (*Search Engine Optimization*) si intendono tutte quelle attività finalizzate a ottenere una migliore accessibilità, architettura e comprensione del sito Web da parte dei motori di ricerca mediante i loro *crawler*, software con il compito di analizzare i contenuti della rete in modo metodico e automatizzato.

SEO è quindi un metodo per strutturare siti e contenuti in modo da ottimizzare la rilevanza e la visibilità del sito d'interesse, garantendone quindi una maggiore esposizione nei motori di ricerca.

In particolare si fa riferimento a due aspetti fondamentali alla base del funzionamento dei motori di ricerca (Fig. 2):

Figura 2:



1. *relevance*, come presa in considerazione delle pagine più rilevanti, nel senso che il motore di ricerca selezionerà i risultati dei siti analizzati attraverso il proprio *crawler* e mostrerà le pagine che meglio si adattano al contesto delle parole chiave (*keyword*) ricercate;
2. *authority*, come grado di affidabilità assegnato a ciascun sito misurato, ad esempio, mediante link esterni collegati al sito di interesse, potendosi considerare un link come voto di fiducia (*vote of authority*).

Il ruolo dei motore di ricerca è quindi quello di individuare l'interrogazione dell'utente all'interno dell'argomento più adatto.

Ne consegue che uno degli aspetti fondamentali della *SEO* risiede proprio nella capacità di misurare questi risultati, andando ad esempio a quantificare il traffico *Web Analytics* organico in modo da valutare il numero degli utenti acquisiti da motori di ricerca.

2.2 CARATTERISTICHE DEI TRACCIAMENTI CON GOOGLE ANALYTICS

Google Analytics [1] è il servizio di *Web Analytics* più usato nel Web.

Dati di w3techs.com [2] del Settembre 2016 evidenziano che ben l'83% dei siti che fanno uso di un sistema di tracciamento utilizzano Google Analytics per la reportistica, e che tale percentuale corrisponde anche al 54.5% dei siti attivi nel mondo.

La gerarchia definita da Google Analytics è così ripartita:

1. vista, come unità più piccola nonché punto di accesso per i report;
2. proprietà, come singolo sito o applicazione;
3. account, come punto di accesso di livello massimo che può contenere una o più proprietà.

I dati raccolti necessitano di essere raggruppati secondo *Dimensioni* e M*etriche* specifiche.

La specifica di dimensioni e metriche definisce quindi l'indicatore chiave di performance (*Kpi*) preso in esame.

Una *Dimensione* è ciò che si vuole misurare all'interno di un rapporto.

Le *Dimensioni* descrivono le caratteristiche degli utenti, delle sessioni o delle azioni compiute dall'utente durante la navigazione. Come esempi:

- paese e città di origine, data e ora;
- comportamento sul sito, ovvero pagine viste, pagina di uscita, ricerca sul sito;
- origine della vista (*mezzo*), se proveniente da motore di ricerca (traffico organico), traffico pubblicitario, sociale, diretto e perfino se proveniente da altro sito.

Una *Metrica* rappresenta il metodo atto a misurare un elemento, o dimensione. Come esempi:

- analisi del traffico, come numero di sessioni, di visitatori etc.;
- qualità della visita, come numero di pagine per sessione, tempo sul sito, frequenza di rimbalzo (*bounce rate*);
- conversioni, come numero di transazioni o di obiettivi raggiunti.

Oltre ad un insieme di *Metriche* e *Dimensioni* standard, è possibile definire delle personalizzazioni associate a dati appositamente ricamati sulle peculiari esigenze del proprio business. L'isolamento e l'analisi di un sottoinsieme di dati specifici viene definita segmentazione.

Tutte queste informazioni sono inviate ai server di Google, in attesa di essere processate.
Google controlla ogni singola porzione del dato come una interazione (*hit*), ed ogni volta che l'utente visita una nuova pagina del sito, il sistema raccoglie e condivide le nuove informazioni in merito alla *custode journey*. L'analisi delle informazioni, che porta alla trasformazione di dati grezzi in statistiche, può comportare dalle 4 alle 24 ore di attesa. Una volta consolidato il dato è possibile ottenere le informazioni attraverso le API di Google Analytics.
Il campionamento (*sampling*) può essere presente nella reportistica di Google Analytics di un sottoinsieme di dati relativi al traffico che, non essendo gestibile in quanto generato casualmente, ne può compromettere la veridicità. Questo perché il campionamento diviene automatico quando l'analisi coinvolge più di 500.000 sessioni per rapporto, cosa che consente al sistema di generare rapporti più rapidamente per tali set di dati.
Si noti come il campionamento dei risultati cresca all'aumentare del periodo di tempo selezionato.
Due sono le strategie possibili per evitare il *sampling*:

- mutuare l'account a Google Analytics 360 [5] potendo contare sulla maggior potenza di calcolo (soglia di *sampling* superiore a 5 milioni di sessioni per rapporto);
- dividere le query di analisi in modo da raggiungere la granularità massima consentita per evitare il sampling.

Per tracciare un sito Web, Google Analytics usa una piccola porzione di codice JavaScript (*tag*) che, inserita in ciascuna pagina del sito, prende vita ed inizia ad accumulare informazioni ad ogni visita dell'utente. I dati raccolti forniscono quindi un preciso riscontro dell'esperienza utente sul sito.
Aspetto impattante dell'installazione del *tag* di Google Analytics in pagina, risiede nella difficoltà che la figura del *Web Analyst* incontra per avere accesso diretto al codice sorgente del sito del cliente.

I sistemi di *tag management* sono stati appunto creati per semplificare l'implementazione e il mantenimento dei tag di Marketing e Misurazione (Google Analytics, Adobe Analytics …).

Il funzionamento di questi sistemi si basa su una singola porzione di codice contenitore (*container*) che, inserita nel sito Web, permette di eseguire tutti gli altri script (*tag*) che vengono gestiti direttamente dall'interfaccia del software, in modo dinamico e sulla base di regole specifiche.

I principali vantaggi derivanti dall'uso di un sistema *tag management* (Google Tag Manager, TagCommander, Tealium, per citarne alcuni) risiedono in:

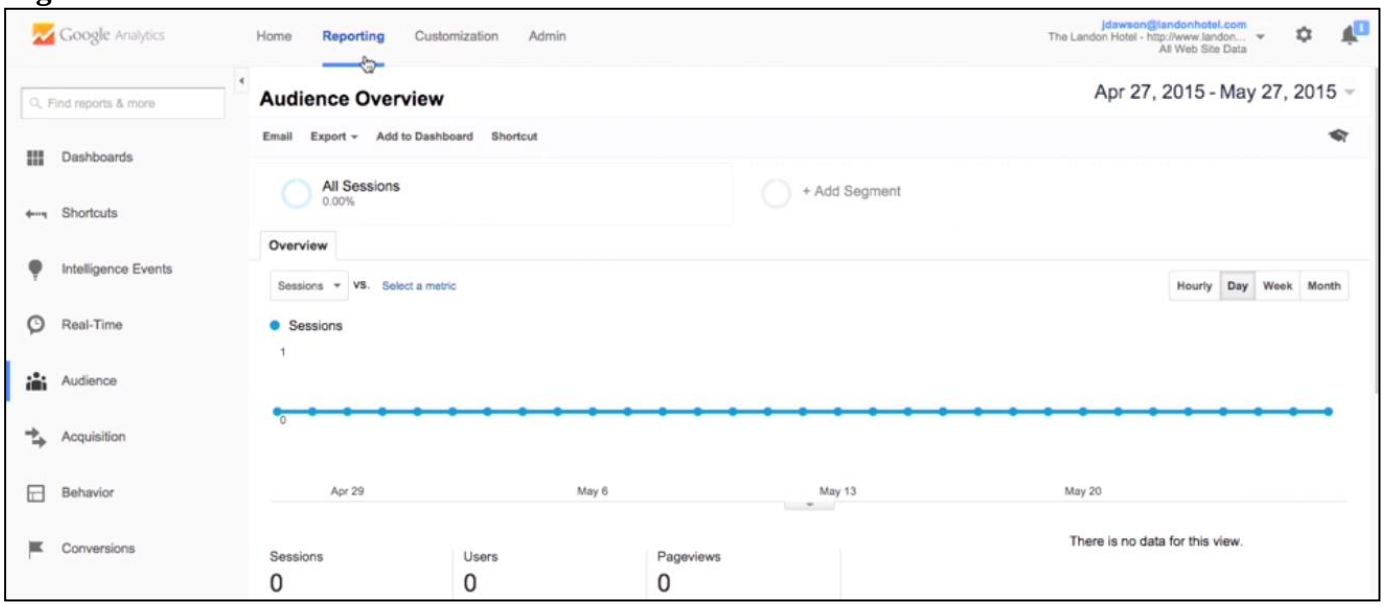
- interventi IT estremamente ridotti;
- debug incorporato;
- flessibilità e modificabilità;
- rapidi tempi di risposta dovuti al caricamento asincrono del *container*.

Entriamo ora nel merito del funzionamento della reportistica di Google Analytics.

Una volta autenticato nella applicazione, si accede alla sezione evidenziata di Reporting.

Come visibile nell'immagine inserita (Fig. 3), la pagina mostra tutti i dati relativi all'account e alla vista selezionata. Sulla sinistra è presente il menu di navigazione, che dispone di varie categorie ulteriormente suddivise in sottocategorie a maggior specificità.

Figura 3:



Tra le categorie specifiche per la reportistica, vi sono:

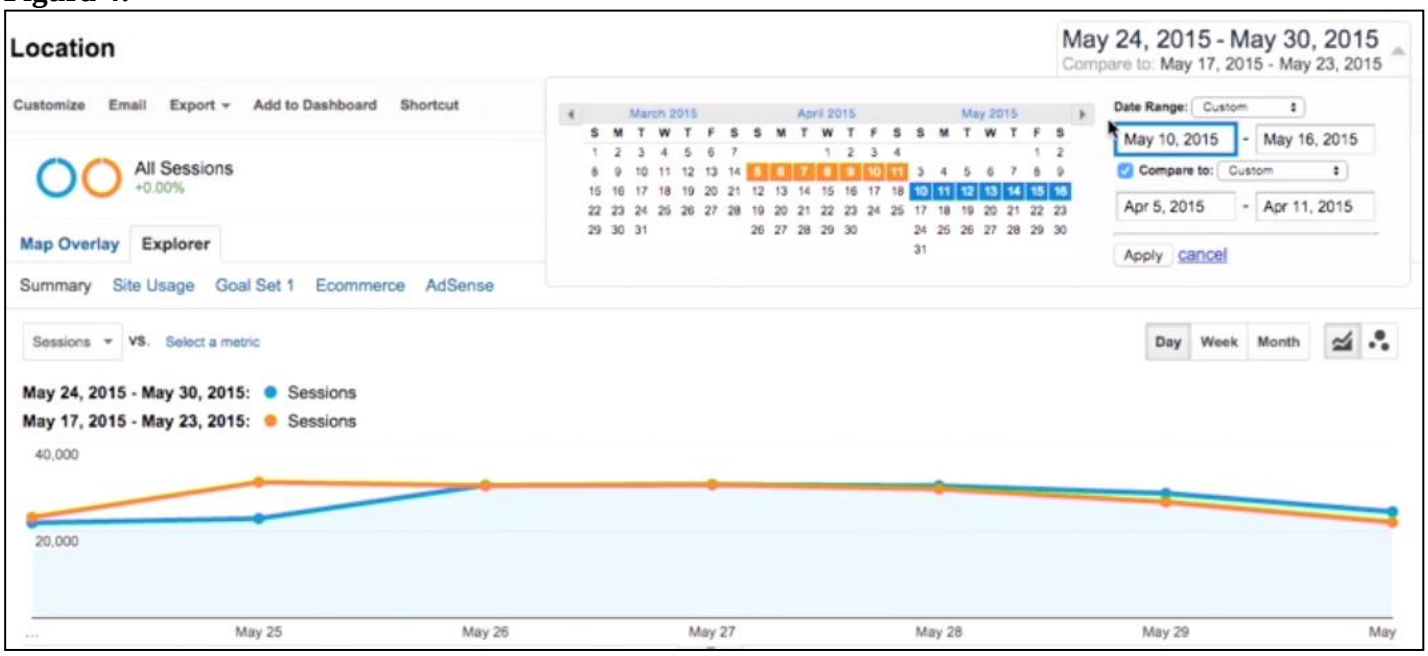
- *Audience*, come report specifici sugli utenti, (dispositivi utilizzati, interessi, geolocalizzazione, etc.);

- *Acquisition*, come report riguardo all'acquisizione di nuovi utenti, potendone analizzare ad esempio il flusso di arrivo sul sito;
- *Behaviur*, come report inerenti l'interazione degli utenti sul sito;
- *Conversion*, come report di analisi accurata delle modalità di funzionamento dei canali di acquisizione atto a portare l'utente verso goal specifici e vendite.

Selezionando *Audience* si può avere accesso a *Active Users*, ma anche alla sottosezione più dettagliata denominata *Demographics*, e questo rende possibile sia uno sguardo generale (*Overview*) sia uno più specifico che comprende ad esempio l'età o il sesso degli utenti tracciati sul sito.

In ultimo cito la funzionalità di comparazione (*compare*) tra distinti archi temporali di analisi, che consente ad esempio un raffronto su base settimanale o mensile. Selezionando un periodo di tempo, tipo dal 24 al 30 Maggio, risulta quindi possibile il confronto di tali dati con quelli inerenti lo stesso periodo (*previous period*) nel mese precedente (Fig. 4).

Figura 4:



2.3 CARATTERISTICHE DEL RANKING SUI MOTORI DI RICERCA

Prima di entrare nel vivo delle tecniche utilizzate volte al miglioramento dei posizionamenti (*ranking*) dei siti sui motori di ricerca, voglio focalizzare l'attenzione sulla cosiddetta *SERP* (*Search Engine Results Page*), pagina dei risultati del motore di ricerca.

Caratteristica comune alla maggior parte dei motori di ricerca concerne i risultati ottenuti da advertising, estremamente differenti da quelli tradizionali (organici) di interesse per la *SEO*. Tali pubblicità sono investimenti derivanti da applicazioni quali *Google AdWords* o Microsoft Bing Ads, che permettono all'inserzionista di scommettere (*bid*) al fine di mostrare le pubblicità nei risultati delle ricerche.

La pagina caratteristica mostra dieci risultati di tipo organico collegati a differenti pagine Web. Ogni risultato, che può anche apparire diverso, deve però nel suo complesso mostrare titolo (*headline*), descrizione, e URL visibile di riferimento.

Un aspetto importante verificatosi a seguito dell'evoluzione di Internet risiede nel profondo cambiamento del contenuto dei risultati, che va ben oltre il testo e le pagine Web, potendosi visualizzare video, notizie, prodotti, mappe, meteo, etc.

La ricerca delle parole chiave (*keyword*) corrette su cui focalizzare la SEO può risultare intricata; le *keyword* sono ciò che gli utenti digitano sui motori di ricerca.

Un soddisfacente piano di ricerca deve contemplare un approccio strutturato volto alla scoperta delle *keyword* presenti nel contenuto del sito di riferimento.

Con i miliardi di interrogazioni effettuate ogni mese, si è resa di fondamentale importanza l'individuazione dei goal di interesse nella ricerca delle parole chiave, in relazione anche alle modalità di raccolta ed analisi dei dati al fine di poter prendere decisioni strategiche sul sito.

In generale la fase di ricerca delle *keyword* può essere riassunta come segue:

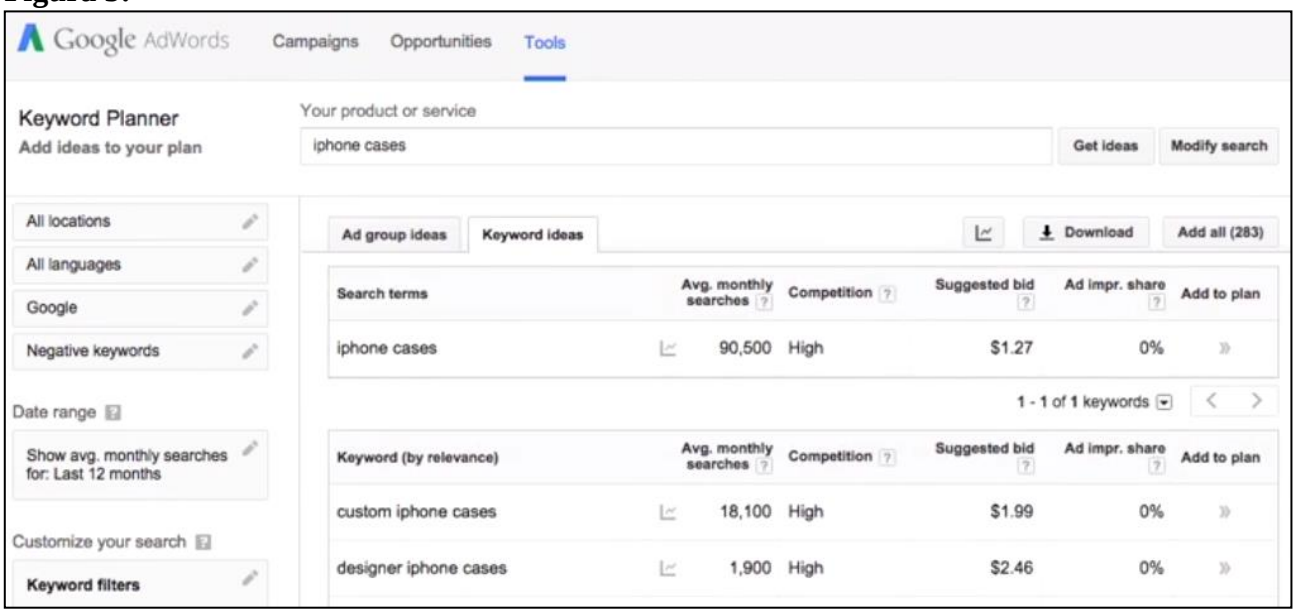
1. brainstorming volta a comprendere ciò che l'utente vuole → esempio una compagnia aerea "low cost" dovrebbe preferibilmente puntare sull'ottimizzazione di parole chiave riguardati le "basse tariffe", piuttosto che sul più generico "servizio aereo di alto valore". Esistono inoltre diversi strumenti per ottenere un elenco di keyword affini a partire da una singola parola chiave;
2. raccolta dei volumi di ricerca, intesi come volume totale delle ricerche degli utenti *keyword* per *keyword*, cercando anche di categorizzare le *keyword* per macro e microargomento.

In particolare lo strumento maggiormente utilizzato per ottenere misurazioni sui volumi di ricerca è Google Keyword Planner [3], per il cui accesso si necessita di un account *Google Adwords* con almeno una campagna pubblicitaria attiva.

Questo strumento pone forte accento su volume, grado di competitività e offerta suggerita per creare un annuncio pubblicitario tramite *Google Adwords*.

Nel menu a sinistra della pagina (Fig. 5) si possono applicare filtri riguardanti tipo di ricerca, geolocalizzazione, ora di ricerca, e categoria.

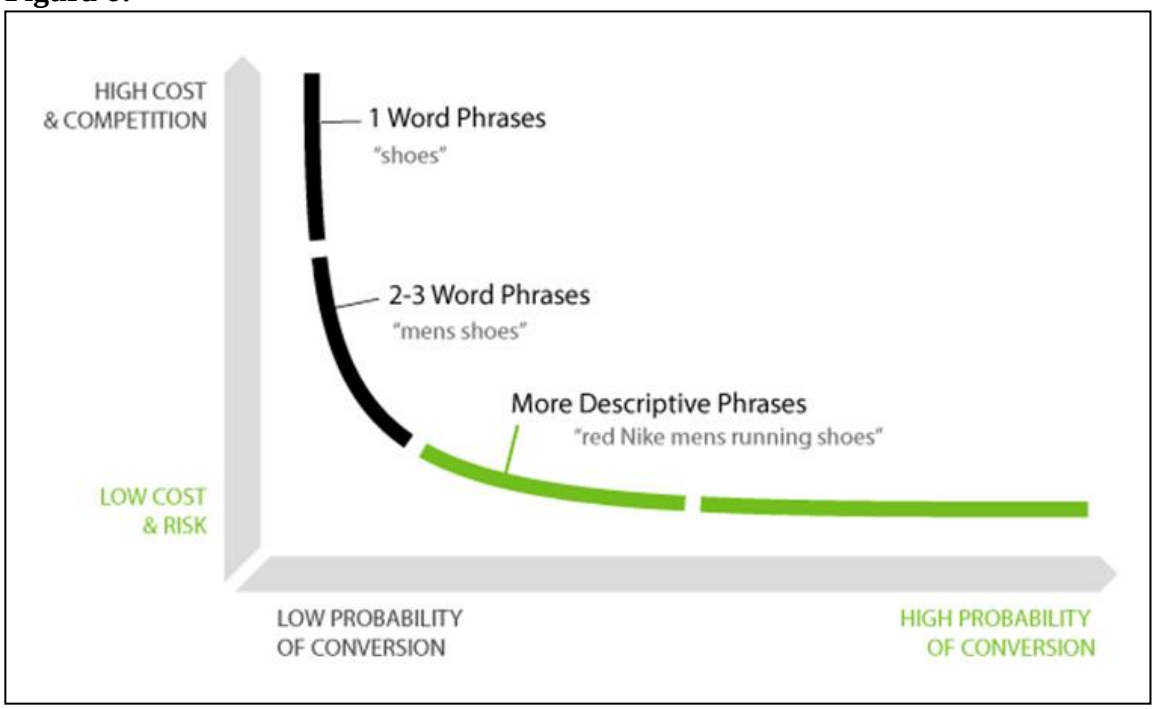
Figura 5:



La ricerca delle keyword da analizzare deve porre particolare attenzione alle cosiddette *keyword* a coda lunga, ovvero parole chiave altamente specifiche (come esempio “custodia iPhone 5s blu”, piuttosto che la più generica “custodia iPhone”).

Questa migliore focalizzazione permette di raggiungere ottimi risultati con maggior probabilità, specie quando la vendita riguarda prodotti molto popolari (Fig. 6).

Figura 6:



Esistono una serie di tecniche universalmente accettate per rendere un sito maggiormente disposto ai motori di ricerca (*search engine friendly*):

- codice HML ben strutturato, che non contenga barriere all'indicizzazione del sito da parte dei motori di ricerca;
- contenuto del sito marcato con tag HTML Meta, in modo da estendere il significato del contenuto e dare rilevanza alla pagina;
- numero di link inbound (diretti al sito di riferimento) e outbound (uscenti dal sito di riferimento) estremamente importante, con particolare rifermento ai termini o all'ambito collegato a ciascun link;
- focalizzazione sul rilascio di contenuto consistente, che deve essere chiaro.

Man mano che le strategie sono implementate, si rende necessario il controllo automatico dei posizionamenti delle *keyword* su cui è stato focalizzato il lavoro di *SEO*, controllo che, come accennato, può essere piuttosto intricato essendo richiesta l'analisi delle pagine dei posizionamenti di ciascuna keyword passo dopo passo.
L'analisi delle *SERP* è estremamente costoso in termini di tempo e risorse, e richiede una grande accuratezza nel salvataggio delle informazioni dal momento che i posizionamenti variano continuamente e non è quindi possibile ottenere *SERP* passate.
Esistono tuttavia diversi strumenti automatici che forniscono i dati dei posizionamenti, e tra i tanti Webranking ha scelto Authority Labs [4]
Il servizio si classifica tra i meno costosi, dato che la richiesta dei dati per *keyword* costa all'incirca 0.0012$ soltanto, e tra i più completi potendo operare sui motori di ricerca maggiormente popolari (Google, Bing, Yandez e Baidu). Inoltre come per il caso di Google Analytics, i dati possono essere scaricati tramite richieste alle API del servizio.

In linea generale è importante comprendere che la *SEO* è un processo continuo a lungo termine, che produce valore nel lungo periodo.
Basti considerare che i motori di ricerca non interagiscono con i siti Web immediatamente, essendo necessario del tempo affinché siano scoperte novità nel contenuto, nei link indirizzati al sito di riferimento e, complessivamente, nei cambiamenti della struttura del codice HTML delle pagine. Questi fattori vanno considerati in relazione anche al tempo richiesto agli algoritmi dei motori di ricerca per il ricalcolo di *relevance* e *authority*, prima che i suddetti cambiamenti siano evidenziati nei risultati delle ricerche.
Risulta quindi cruciale comprendere che non si può mai avere il completo controllo del procedimento.

Come se non bastasse è necessario tener conto del fatto che gli algoritmi dei motori di ricerca sono in continua evoluzione, pertanto gli effetti di questi cambiamenti sulle *SERP* non possono essere sempre prevedibili.

Tuttavia è bene considerare che il lavoro di ottimizzazione *SEO* è rivolto al raggiungimento del massimo risultato possibile di due audience contemporaneamente, motore di ricerca e utenti.

In particolare la vera audience, quella che genera i miglioramenti più importanti, è costituita dalle persone.

3 DEFINIZIONE E ANALISI DEL PROBLEMA

La sezione presenta e definisce il problema trattato, problema che ha spinto appunto alla progettazione della *VAD*.
Il primo passaggio, logico ma anche temporale, consiste nella definizione del caso di studio da prendere in esame.
I punti fondamentali sono tre:

1. descrivere un problema in termini tali da renderlo risolvibile;
2. formulare delle ipotesi sulle possibili cause del problema;
3. cercare di esporre il caso di studio non in termini di possibili soluzioni, ma focalizzando l'attenzione sulle conseguenze negative.

Molti sono i casi in cui l'attività di progettazione stessa, mossa da un problema, consiste proprio nella individuazione delle modalità di risoluzione dello stesso, definendo le azioni necessarie per la realizzazione della soluzione.
Non bisogna dimenticare che, spesso e volentieri, le problematiche scaturiscono proprio dalle specifiche esigenze imposte dai clienti committenti, richieste che necessitano di essere relazionate all'efficienza e ai limiti delle sorgenti per il recupero del dato.

3.1 CASO DI STUDIO

Due sono i clienti che hanno sottoposto il problema a Webranking.
La Acme e il gruppo Oceanic Airlines sono compagnie internazionali altamente articolate e tale complessità si riflette anche nelle aree di *Web Analytics* e *SEO*.
Richiesta imperativa dei committenti risiede nella facoltà di visualizzare diversi *Kpi* appartenenti ad entrambe le aree grazie a due viste di presentazione principale con le informazioni sintetiche da un lato e quelle puntuali dall'altro.
Il progetto ha quindi organizzato i dati tra pagina principale e lista di dettaglio (una per *Kpi*) al fine di poter filtrare, aggregare e confrontare dati in base ai siti ripartiti per paese d'origine e sotto-brand di ciascun cliente.
Per questa ragione si è quindi deciso di sviluppare la *VAD*.

Per definizione una dashboard è uno strumento interattivo di raccolta, monitoraggio e visualizzazione di dati e informazioni.
L'obiettivo primario della dashboard progettata si focalizza sull'aggregazione e la visualizzazione dei dati appartenenti a più di 250 siti diversi dei clienti.

3.2 DASHBOARD MULTI CLIENTE E MULTI UTENTE

Caratteristica imprescindibile è la gestione lato utente-cliente.
Per ragioni commerciali, i dati contenuti nei siti di entrambi i clienti hanno la necessità di essere ripartiti su account diversi, non essendo possibile la condivisione di dati in relazione, soprattutto, ai riscontri degli investimenti di *Web Advertising* effettuati.
Ricordo inoltre come i siti di riferimento di ciascun cliente non siano soltanto molteplici, ma anche ripartiti su diversi sotto-brand del gruppo.
L'analisi del requisito ha indotto l'implementazione di una associazione del tipo "uno a molti" tra utente e clienti (brand). In tal modo è stato possibile appaiare ciascun utente a un sottoinsieme di clienti, garantendo così una chiara suddivisione e ripartizione dei dati all'interno della medesima applicazione.

3.3 INTEGRAZIONE DI TIPI DI DATO ETEROGENEI

Facendo riferimento alle caratteristiche proprie sia del traffico *Web Analytics*, che dei posizionamenti delle keyword *SEO*, appare subito evidente la non omogeneità dei dati come visibile in tabella.

Confronto tra Kpi Web Analytics e SEO

Kpi	Metriche	Dimensioni	Keyword	Volume	Posizionamento
Sessions	ga:sessions	ga:date ga:deviceCategory	- -	- -	- -
Keyword Ranking	- -	- -	keyword1 keyword2 keyword3 keyword n.	volume1 volume2 volume3 volume n.	posizionamento1 posizionamento2 posizionamento3 posizionamento n.

Se da una parte si hanno i dati a cadenza giornaliera di particolari *Kpi* con metriche e dimensioni differenti, dall'altra è presente un elenco di keyword, volumi e posizionamenti relativi al sito di appartenenza.

La prima riga della tabella mostra il *Kpi Web Analytics*, denominato *Sessions*, comprendente i dati delle sessioni degli utenti ripartiti sulle dimensioni di data (*ga:date*) e categoria di dispositivo utilizzato (*ga:deviceCategory*). Si noti che tale *Kpi* non presenta informazioni tipo *Keyword*, *Volume*, e *Posizionamento*, che ricordo essere invece i tipici dati relativi alla *SEO*.
Proseguendo con il secondo *Kpi* denominato *Keyword Ranking*, relativo ai volumi e ai posizionamenti di un elenco di *keyword*, appare subito evidente la completa mancanza di informazioni nelle due colonne *Metriche* e *Dimensioni*, che appartengono infatti ai soli *Kpi* di tipo *Web Analytics*.

In seguito verranno mostrate le differenze specifiche di questo ultimo insieme di *Kpi* ed anche il modo in cui questa peculiare caratteristica abbia profondamente guidato l'implementazione dell'applicazione.

3.4 INTEGRAZIONE DI API RIFERITE A BRAND E PAESI DIVERSI

Una grande limitazione imposta dalla versione gratuita di Google Analytics risiede nell'impossibilità di aggregare e confrontare dati relativi a viste (siti) differenti.
Il problema, facilmente aggirabile navigando su due pagine riferite a viste distinte, non può tuttavia essere "risolto" nella situazione in cui i siti appartengono ad account diversi che richiedono pertanto autenticazioni distinte.
La gestione dei dati relativi al notevole numero di siti raccolti ha portato a considerare i possibili strumenti alternativi presenti al momento sul mercato.

3.4.1 Google Analytics 360

In prima battuta è stata valutata la possibilità di aggiornare Google Analytics 360, che permette una aggregazione multi-vista.
La tabella di seguito riportata ne mostra le differenze funzionali con la versione *Standard*, gratuita.
Tale soluzione alternativa all'auto-sviluppo interno è stata scartata in considerazione delle eccessive caratteristiche aggiuntive, tenendo soprattutto conto del risultante costo annuale pari a ben 150.000 $.

Confronto funzionalità tra *Google Analytics Standard e Google Analytics* 360

Funzionalità	Standard	Google Analytics 360
Hit mensili	10.000.000	1.000.000.000
Campionamento sessioni	fino a 500.000	fino a 25.000.000
Report non campionati	NO	SI
Roll-Up Reporting (esclude il traffico proveniente da client e fornitori in modo da considerare solo gli utenti "puri")	NO	SI
Integrazione con DoubleClick Campaign Manager (per la gestione delle campagne di advertising)	NO	SI
Modello di attribuzione basato sui dati raccolti	NO	SI
Dimensioni personalizzate	20	200
Metriche personalizzate	20	200
Metriche calcolate	NO	SI
Tabelle personalizzate	NO	SI
Funnel personalizzati (*customer journey* che porta l'utente alla conversione)	NO	SI

3.4.2 Tableau

Successivamente si è preso in considerazione Tableau [6], software di business intelligence che consente di collegare sorgenti di dato e di creare dashboard interattive.
Tra le caratteristiche del prodotto si annoverano:

- usabilità;
- scalabilità;
- alta disponibilità;
- supporto a sorgenti di dato eterogenee;
- supporto mobile nativo;
- possibilità di visualizzazione offline e real time;
- condivisione di compiti tra utenti con flussi di lavoro comuni.

Secondo quanto riportato sul sito di riferimento, Tableau risulta essere attualmente leader tra le piattaforme di Business Intelligence e Analytics, così come nei quattro anni precedenti (2012-2016), potendo contare su ben 35.000 account, peraltro in continua crescita.

3.4.2.1 Caratteristiche

Tableau basa il suo potente sistema di Analisi sulle informazioni estrapolate dalle sorgenti di dato (*data source*), che vengono mostrate successivamente in dashboard altamente personalizzabili.
Un *data source* consiste di meta dati che descrivono:

- connessioni a dati multiple, che definiscono il tipo di dato da analizzare con Tableau;
- personalizzazione e filtro per consentire operazioni quali calcoli, raggruppamenti, formattazioni personalizzate, etc.;
- differenti soluzioni di aggiornamento del dato.

La gestione delle sorgenti di dato online comprende:

- permessi assegnati a specifici utenti o gruppi quali connessione, modifica o scaricamento di *data source*;
- operazioni programmate di aggiornamento del dato.

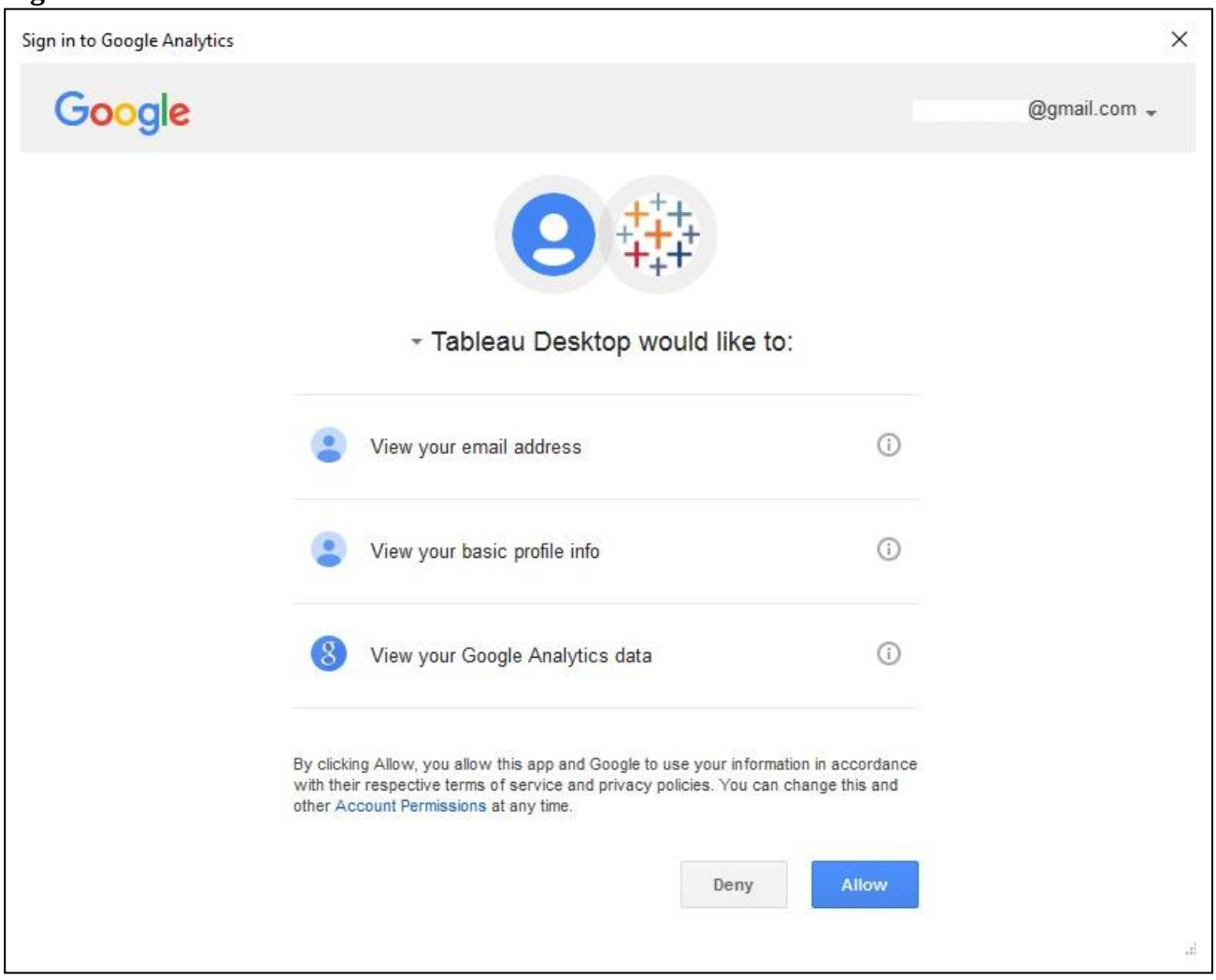
Una dashboard così creata comprende una collezione di differenti fogli di lavoro (*worksheet*), che possono mostrare diverse sorgenti di dato contemporaneamente.

3.4.2.2 Supporto a Google Analytics

Tableau supporta nativamente l'integrazione con dati provenienti da Google Analytics, che consistono in semplici collegamenti ad estratti di *data* source, con configurazione definita da autenticazione Google.

Per abilitare questa funzionalità è richiesto il permesso utente (Fig. 7).

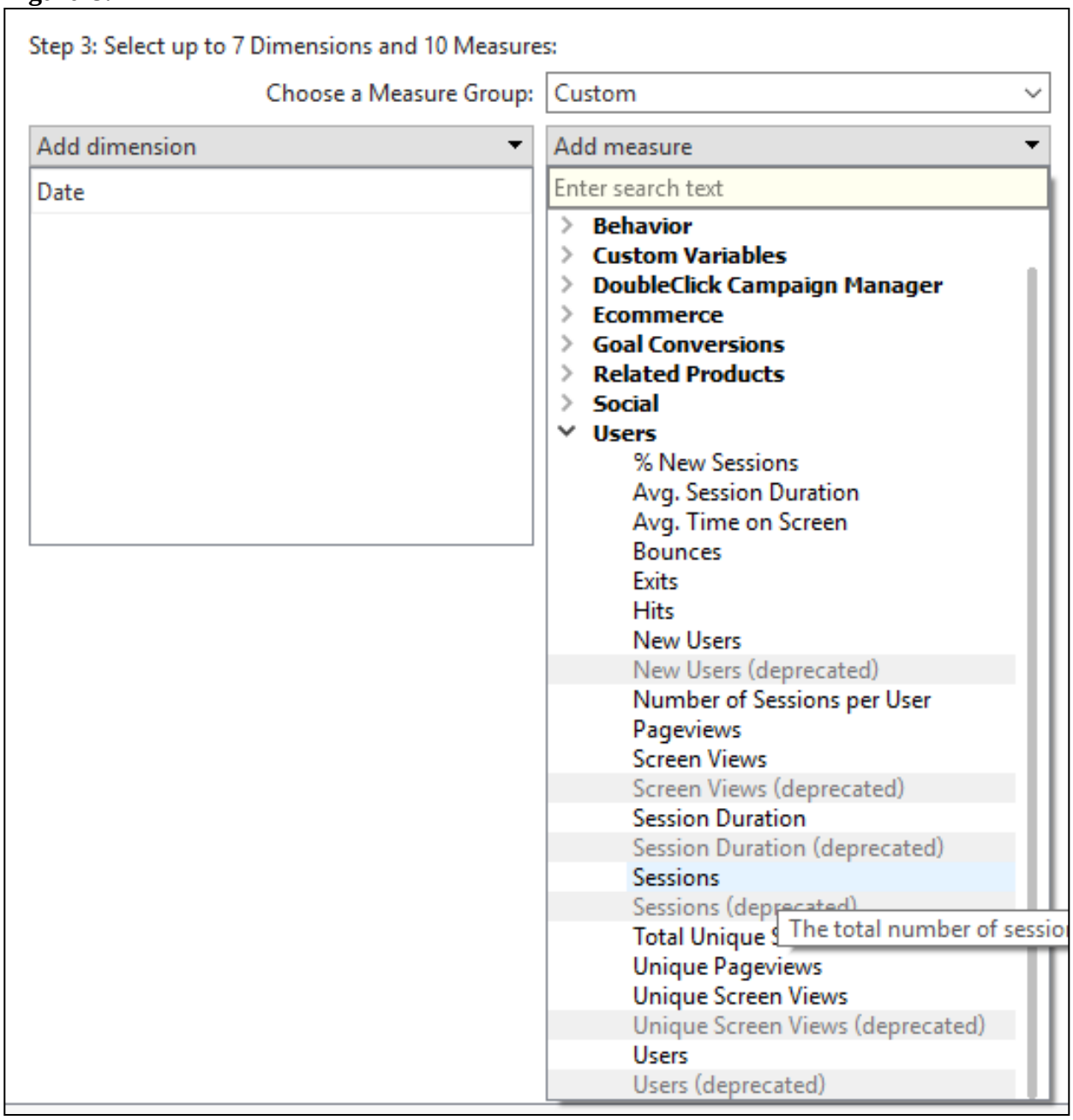
Figura 7:



In presenza di un minor numero di *data source* (configurazione considerata "standard"), Tableau si interfaccia facilmente permettendo di cercare e selezionare *Metriche* e *Dimensioni* per definire singoli *Kpi*.

Questa caratteristica consente rapide analisi, quando l'obiettivo è quello di valutare *Kpi* diversi presi individualmente (Fig. 8).

Figura 8:



3.4.2.3 Criticità

Come accennato, caratteristica imprescindibile è stata l'aggregazione dei dati relativi a siti distinti, suddivisi su più sotto-brand di ciascun cliente. In aggiunta si evidenzia la necessità di configurare *data source* differenti, per ciascun *Kpi* che non condivide *Metriche* e *Dimensioni*.

Al fine di replicare le caratteristiche richieste, lo studio di fattibilità si dovrebbe avvalere della configurazione di oltre 650 *data* source differenti, con il risultato di insormontabili difficoltà. Lo stesso supporto interno di Tableau ci ha confermato che non è proprio possibile la gestione di una tale problematica.

Inoltre non è stato possibile comprendere se Tableau consenta o meno la gestione contemporanea di un così alto numero di sorgenti del dato, senza considerarne poi il successivo mantenimento ed evoluzione che si tradurrebbe in un grande sforzo di riconfigurazione.
Tornando alla gestione dei *Kpi*, si evidenzia che la complicata gestione di *Metriche* e *Dimesioni* incompatibili può portare alla raccolta di risultati vuoti, o ancor peggio sbagliati.
Si consideri per esempio:

- Sessions, come aggregato del numero di sessioni totali:
  - Metriche: ga:sessions;
  - Dimensioni: ga:date, ga:deviceCategory.
- Traffic Organic and Paid, come aggregato delle sessioni derivanti da traffico organico e pubblicitatio:
  - Metriche: ga:sessions;
  - Dimensioni: ga:adDistributionNetwork, ga:date, ga:deviceCategory, ga:medium.

Le dimensioni relative a "Traffic Organic and Paid" sono più restrittive se confrontate con i risultati ottenuti da "Sessions". Ne segue che il numero totale del secondo *Kpi* in elenco risulta maggiormente filtrato, se comparato con il numero puro di sessioni.
La risoluzione del problema porta ad una configurazione di *data* source ad hoc pari al numero esatto di *Kpi* con *Metriche* e *Dimensioni* incompatibili.

3.4.3 Dashboard Personalizzata

Visti i limiti derivanti dalla complessità delle richieste, si è proceduto ad una analisi comparativa delle differenze tra l'utilizzo di un software avviato come Tableau, rispetto alla creazione di una apposita applicazione Web-based personalizzata.
Webranking ha verificato la fattibilità in relazione agli obiettivi dei clienti, contattando anche il supporto ufficiale Tableau in modo da assicurare che la strada scelta fosse davvero la più adatta a risolvere i problemi e a raggiungere gli obiettivi preposti.
I risultati di questo studio, riassunti nella tabella inserita nella pagina successiva, evidenziano che Tableau non si è rivelato del tutto adeguato al caso specifico, vista la complicata configurazione iniziale derivante dall'alto numero di *data source* richieste.
Particolare attenzione va inoltre posta al mantenimento del dato, che può risultare estremamente intricato e sensibile all'errore.

Confronto tra Tableau e Dashboard personalizzata

Caratteristiche	Tableau	Visual Analytics Dashboard
Configurazione iniziale (considerando 250 siti)	✗	✗✗✗
Aggregazione di *Kpi* a partire dale viste	✗	✗✗✗
Divisione del dettaglio del dato a diversa granularità	✗✗	✗✗✗
Divisione del dato sito per sito	✗✗✗	✗✗✗
Divisione del dato per dispositivo di origine del traffico	✗✗✗	✗✗✗
Aggiunta di nuovo sito per ciascun *Kpi*	✗	✗✗✗
Comparazione per data	Non è stato trovato modo di implementare la funzionalità	✗✗✗
Comparazione per sito	✗✗✗	✗✗✗
Comparazione tra gruppi di siti	✗	✗✗✗
Analisi veloce su singolo *data source*	✗✗✗	✗
Portabilità	✗✗	✗✗✗
Integrazione con altre piattaforme	✗✗✗	✗✗✗
Tempi di caricamento	✗✗✗	✗✗✗

Si è quindi proceduto alla creazione della applicazione denominata Visual Analytics Dashboard (*VAD*), la cui progettazione e realizzazione troverà spazio nel paragrafo successivo.

3.5 KPI PERSONALIZZATI ALLE DIVERSE ESIGENZE

Facendo riferimento alle caratteristiche proprie del traffico *Web Analytics*, si ricorda la necessità di creare *Kpi* personalizzati alle esigenze dei clienti.

Si pensi ad esempio alla creazione di un *Kpi* relativo all'azione di "Download della Brochure" da uno o più siti Web del cliente, caso che prevede due alternative nella configurazione delle *Metriche* e *Dimensioni*:

1. configurazione ad hoc del *tag* di Google Analytics all'interno della pagina in cui è presente il link per il download della brochure → creazione di goal Google Analytics personalizzati faciliterà l'aggregazione dei dati;
2. definizione specifica dei valori di *Metriche e Dimensioni* desiderati → creazione di un filtro sulla *Dimensione* ga:eventCategory per i soli eventi relativi al modello "download_brochure".

3.6 MANTENIMENTO DEL DATO VS. SCARICAMENTO IN TEMPO REALE

Per quanto riguarda il tema legato alla rappresentazione del dato, si è valutata la scelta tra due approcci metodologici di analisi dati contrastanti, *Web Analytics* e *SEO*, rispettivamente provenienti dalle richieste API dei servizi Google Analytics e Authority Labs.

Da una parte lo scaricamento in tempo reale tramite combinazione di richieste API effettuate al momento, dall'altra l'approccio basato sul salvataggio automatizzato del dato a database.

Con particolare riferimento ai tracciamenti di Google Analytics, si evidenzia che la scelta di scaricamento in tempo reale porta ovviamente a non avere discrepanze nei valori dei dati caldi restituiti tra lato dashboard e piattaforma sorgente.

Tuttavia questo sistema comporta un elevato quantitativo di richieste, corrispondenti al numero siti selezionati moltiplicati per ciascun *Kpi* visualizzato.

In aggiunta per quel che concerne i dati *SEO*, l'approccio real-time sarebbe naturalmente impossibile dal momento che Authority Labs mostra i risultati dei posizionamenti soltanto a seguito di una richiesta POST HTTP volta all'analisi delle interrogazioni stesse.

Da qui la scelta di scaricare e mantenere il dato periodicamente.

Nell'ambito *SEO* si è scelto di aggiornare i posizionamenti delle keyword con cadenza mensile, per un totale di circa 6.000, trattandosi di dati meno sensibili a modifiche nel breve periodo e tenuto conto dei costi di utilizzo delle API di Authority Labs.

Nel caso *Web Analytics* invece, è stata programmata una cadenza giornaliera con l'incognita però di scaricare dati ancora non "assestati", rischio che sarà preso in esame al capitolo 3.8.

3.7 CAMPIONAMENTO

Come accennato, le richieste di dati a Google Analytics può prevedere il campionamento.
Il *sampling* è riscontrato nel caso in cui le richieste di ogni singola porzione di dato (*hit*) superino le 500.000 sessioni per rapporto, provocando la compromissione della veridicità dei dati ottenuti.
Tra le strategie possibili per evitare *sampling* è da escludersi l'aggiornamento a Google Analytics 360, per le ragioni precedentemente esposte. Unica alternativa possibile resta quindi la scissione delle interrogazioni di analisi in modo da raggiungere la granularità massima consentita per evitare l'innescarsi del fenomeno di campionamento.
Questa soluzione, riportata in Appendice, ha permesso quindi di mantenere dati esenti da sampling consentendo aggregazione e confronti puri su archi temporali senza limiti di durata, cosa che un approccio di scaricamento in tempo reale, quale da piattaforma di Google Analytics, non contempla.

3.8 CONSOLIDAMENTO DEL DATO

Lo scaricamento lato *Web Analytics* è stato strutturato decidendo di scaricare ogni giorno i dati del giorno precedente. Questo approccio potrebbe tuttavia causare problematiche in quanto la latenza dichiarata per l'elaborazione dei dati raccolti da Google Analytics può variare tra le 24 e le 48 ore. L'invio di più di 200.000 sessioni al giorno da parte degli account "standard" comporta l'aggiornamento dei rapporti una sola volta al giorno, fatto che può ritardare gli aggiornamenti dei rapporti fino a un massimo di due giorni. Per ripristinare l'elaborazione nell'ambito della giornata, sarebbe dunque necessario limitare l'invio a meno di 200.000 sessioni al giorno oppure, anche in questo caso, l'aggiornamento ad Analytics 360 potrebbe spostare questo limite a 2 miliardi di hit al mese (oltre 66 milioni di hit al giorno).
Avendo precedentemente escluso l'ipotesi di aggiornamento, così come la possibilità di limitare il numero di hit raccolte, si è deciso di implementare una procedura di "*rinfrescamento*" del dato. Questa procedura, riportata in Appendice, prevede che siano riscaricati ogni giorno i dati *Web Analytics* relativi ai 3 giorni precedenti.
Inoltre, per aggirare i bug che influenzano i dati di Google Analytics è stata prevista, con cadenza mensile, una nuova procedura volta al rinfrescamento dei dati del mese precedente (https://code.google.com/p/analytics-issues/issues/list pagina che raccoglie i bug riportati dagli utenti).

4 VISUAL ANALYTICS DASHBOARD PROGETTO ED IMPLEMENTAZIONE

4.1 SOLUZIONI ARCHITETTURALI

Definite a grandi linee le funzionalità principali dell'applicazione *VAD*, entriamo nel dettaglio delle possibili scelte architetturali per giungere alla realizzazione.

4.1.1 Considerazioni sulla scelta architetturale

La prima scelta concerne l'alternativa tra *"make or buy"* dell'infrastruttura hardware e software alla base dell'applicazione.

Si è subito proceduto all'abbandono della soluzione *"make"* in quanto è nostro obiettivo potere scalare l'applicazione agilmente, valutando passo dopo passo le modifiche eseguite. I servizi offerti dal *cloud computing* consentono infatti di regolare le risorse su richiesta, garantendo quindi alta scalabilità.

Effettuata la scelta di affidarsi a servizi esterni in cui l'applicazione dovrà appoggiarsi, ci si è ritrovati di fronte a 3 tipologie di servizio:

1. *Infrastructure as a Service (IaaS)*, dove viene fornita un'intera infrastruttura virtualizzata con possibilità di scegliere tra diversi livelli di potenza di calcolo, spazio disco, servizi di rete, sistema operativo, linguaggi ed extra con i quali poter erogare i propri servizi;
2. *Platform as a Service (PaaS)*, dove viene fornita una piattaforma di sviluppo, test, implementazione e gestione di una applicazione Web;
3. *Software as a Service (SaaS)*, dove viene fornito il software applicativo fruibile online.

Scartata la possibilità di utilizzare *SaaS* per i limiti che questa soluzione impone, si sono confrontate le restanti tipologie di servizio.

Se da un lato *PaaS* fornisce il giusto compromesso tra disponibilità di scelta implementativa e facilità di utilizzazione, dall'altro *IaaS* garantisce la massima libertà di configurazione, e contribuisce nel contempo a una riduzione dei costi non avendo stretta necessità di appoggiarsi a servizi esterni.

In considerazione del numero di servizi richiesti dalla applicazione (Supervisor, MySQL, Mongo, Cron...), si è dunque deciso di utilizzare la tipologia *IaaS* per avere maggiore libertà implementativa.

4.1.2 Amazon Web Services

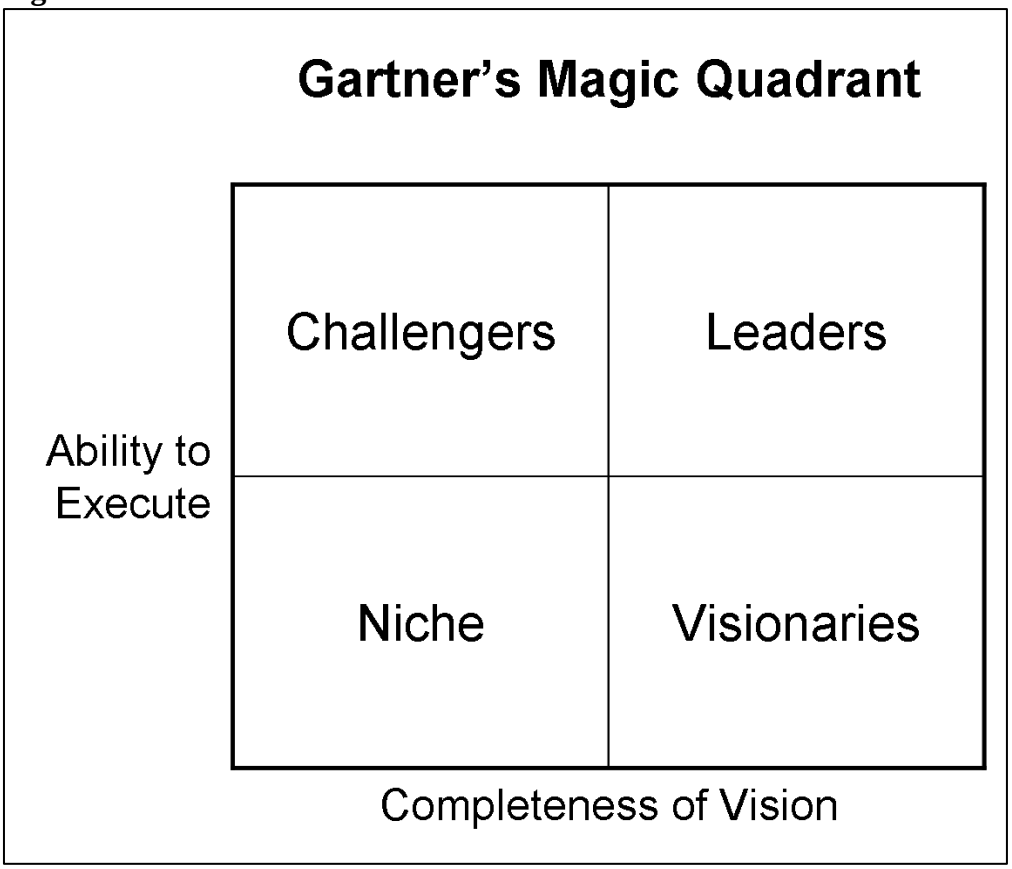
La scelta del servizio di hosting dell'infrastruttura applicativa si è basata sulla ricerca qualitativa di Gartner, denominata Magic Quadrant.

Gartner è una società per azioni multinazionale leader mondiale in consulenza strategica, ricerca e analisi nel campo dell'Information Technology con oltre 60.000 clienti nel mondo, che ha come attività principale quella di supportare le decisioni di investimento dei suoi clienti mediante ricerca, consulenza, benchmarking, eventi e notizie.

I Magic Quadrant forniscono una analisi delle performance dei fornitori appartenenti a un determinato segmento di mercato, con l'obiettivo ultimo di aiutare gli acquirenti ad assumere decisioni più consapevoli nella scelta delle aziende partner che meglio soddisfano le loro esigenze in termini di servizi e prodotti.

Si tratta di un grafico bidimensionale molto intuitivo, dove risulta estremamente semplice individuare come si spostano i prodotti all'interno del diagramma disponendosi in uno dei quattro sotto Quadranti: Leaders, Challengers, Visionaries e Niche Players (Fig. 9).

Figura 9:



Nel Magic Quadrant 2016 per le infrastrutture Cloud *IaaS* (Fig. 10), Gartner ha inserito Amazon Web Services (AWS) [7] nel quadrante dei "Leader" a livello mondiale per il 6° anno consecutivo, posizionando AWS al primo posto per completezza di visione e abilità di esecuzione.

In considerazione di questi risultati visualizzati nella prossima figura, la scelta non poteva che ricadere sulla infrastruttura di Amazon Web Services.

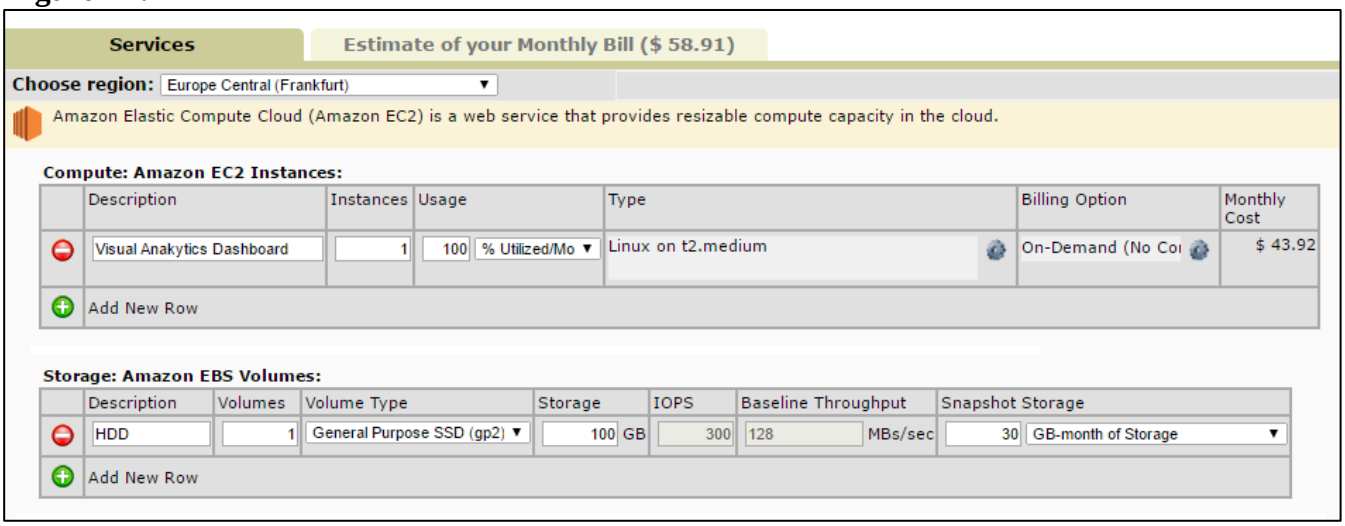
Figura 10:

Il servizio *IaaS* per eccellenza di AWS è rappresentato da Elastic Computing Cloud denominato AWS EC2, dove C2 è un gioco di parole ovvero l'acronimo ECC diviene EC2.
EC2 fornisce un'infrastruttura Web per disporre di macchine virtuali completamente su richiesta (*on demand*) con facoltà di configurare, avviare, spegnere e clonare istanze di macchine virtuali interamente costruite secondo le proprie necessità. Inoltre è possibile avviare immagini (*snapshot*) già pronte e messe a disposizione della comunità con le applicazioni più diffuse.
La scelta della macchina EC2 adoperata per l'implementazione si è basata su un giusto compromesso tra livello di utilizzo e prestazioni richieste.
Da un lato si è considerato il limitato utilizzo della applicazione per quanto riguarda numero di utenti connessi e frequenza di aggiornamento del dato (frequenza massima di un giorno).
Dall'altro si è valutato l'uso massivo delle risorse richiesto per l'aggregazione dei dati dei *Kpi*.
Si è dunque optato per una istanza di tipo T2-Medium, che fornisce un livello base di prestazioni potendo contare su 2 core virtuali, con possibilità di superamento delle performance (*overclocking*) temporanea.
La pagina di documentazione ufficiale [8] asserisce che questo tipo di istanza rappresenta una buona scelta per carichi di lavoro che non sfruttano completamente la CPU in termini di regolarità di utilizzo, ma che necessitano soltanto occasionalmente di un incremento delle prestazioni (ad es. server Web).
Tra le caratteristiche della macchina si riportano:

- 2 CPU virtuali Intel Xeon con Turbo fino a 3.3 GHz;
- 4 GB di memoria;
- 100 GB di storage Amazon Elastic Block Store (EBS);
- Ubuntu 14.04 a 64 bit.

Le prestazioni di base e le relative possibilità di superamento sono gestite tramite i cosiddetti crediti CPU. Ogni istanza T2 riceve continuamente crediti CPU ad una frequenza prestabilita in base alle dimensioni dell'istanza e accumula crediti CPU quando non è attiva, per poi utilizzarli quando necessario.
Di seguito si riporta il dettaglio dei costi del servizio ricavato tramite il calcolatore di prezzo ufficiale (Amazon Price Calculator via [9] che presenta un canone mensile pari a 58.91$ per la macchina scelta (Fig. 11).

Figura 11:

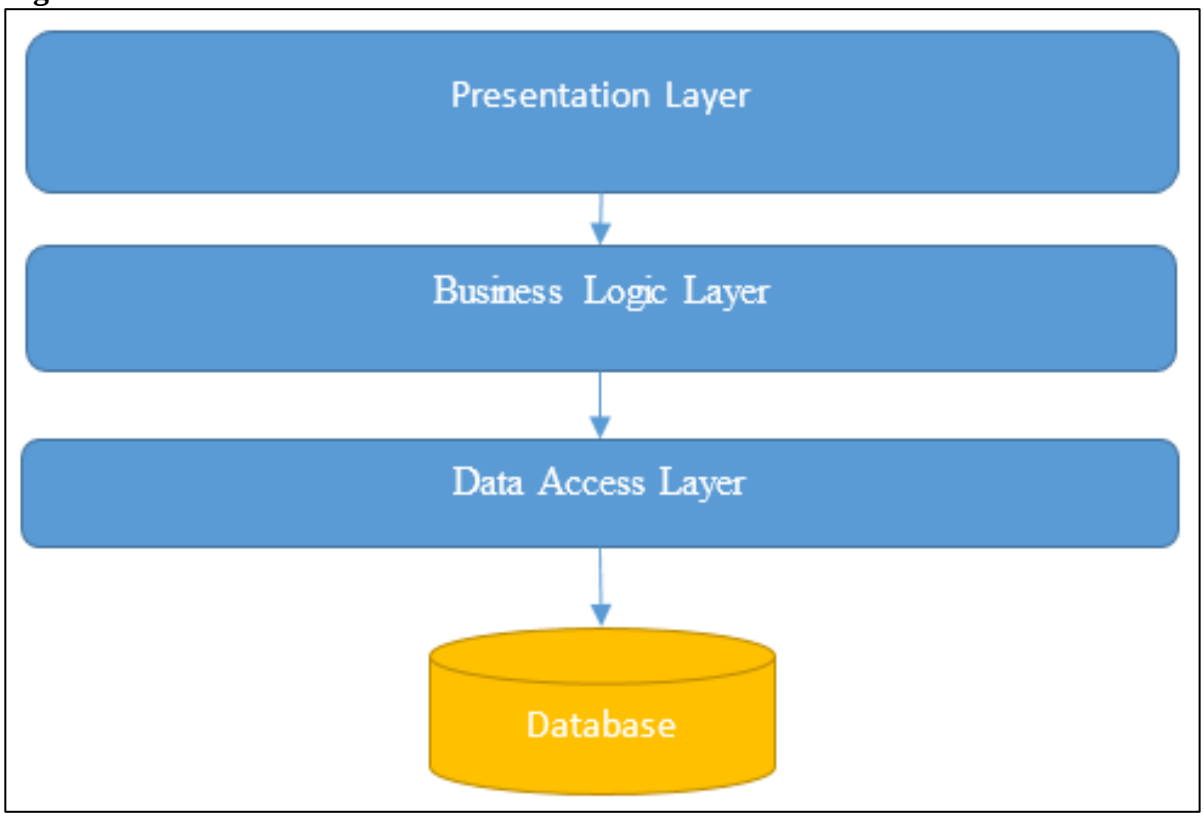


4.2 DEFINIZIONE DEI LIVELLI LOGICI DI PROGETTO

Una applicazione Web moderna presenta una ripartizione in 3 livelli logici (Fig. 12):

1. livello di presentazione (*presentation logic*);
2. processi logico funzionali (*business logic*);
3. archiviazione ed accesso ai dati (*data logic*).

Figura 12:

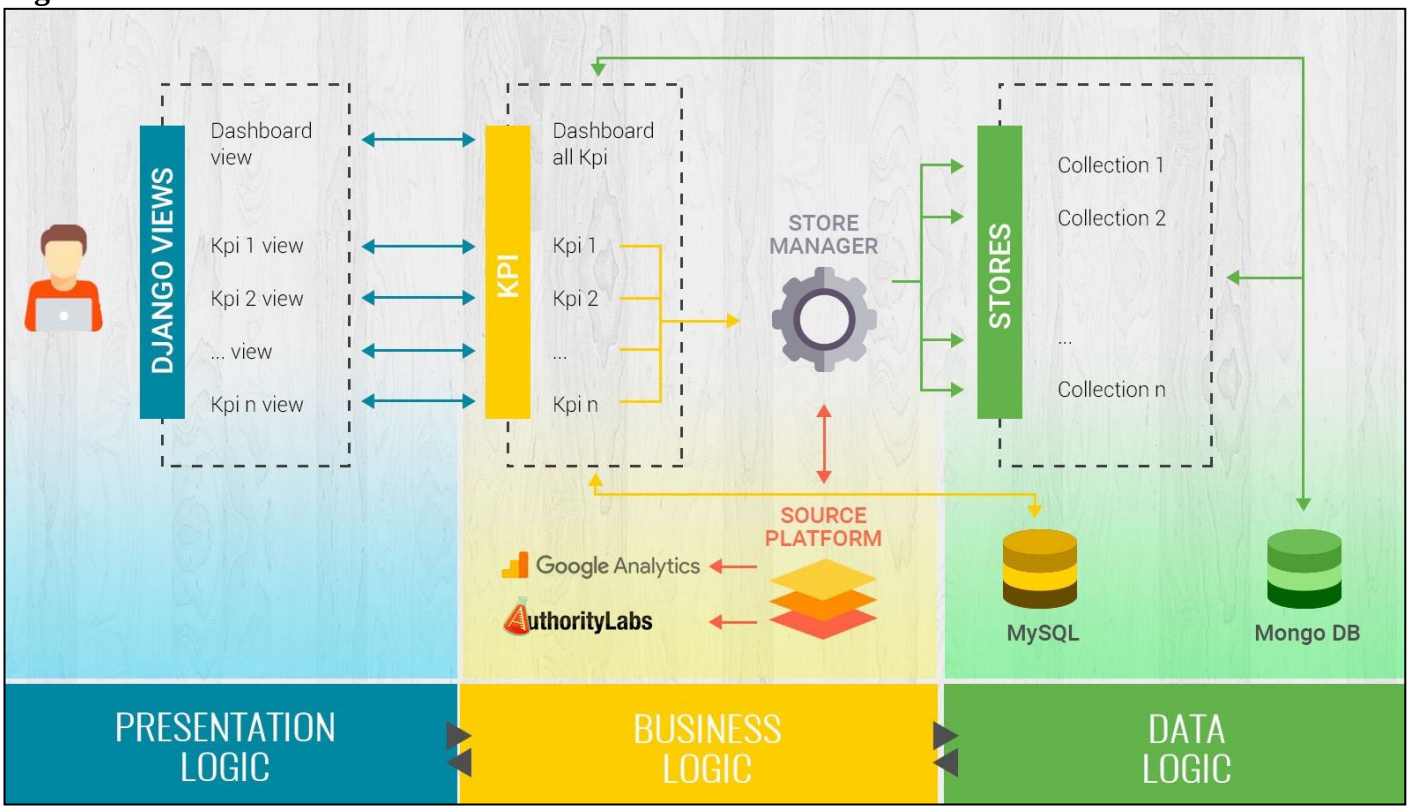


I livelli logici sono sviluppati e mantenuti come moduli indipendenti; da notare che i processi implementano i livelli logici, mentre i computer implementano i processi.

Si è quindi deciso di adottare tale ripartizione in livelli logici.

Di seguito si inserisce lo schema che racchiude il percorso logico della *VAD* (Fig. 13).

Figura 13:



Per quanto concerne il lato frontend, l'utente autenticato ha accesso ad una serie di pagine (*templates*) realizzate in HTML e referenziate al lato backend dell'applicazione tramite viste Django.

A seconda delle vista acceduta sono richiamate una serie di azioni volte al recupero del dato desiderato:

1. Ciascun *Kpi* memorizzato in tabella SQL MySQL è definito dai propri campi (tipo *SEO*, metriche, dimensioni, MongoDB_store…);
2. Lo *Store Manager* si occupa di garantire lo scaricamento e/o l'aggregazione dei dati scelti in base alle caratteristiche del *Kpi*;
3. La *Source Platform* definisce la piattaforma per il recupero dei dati;
4. Lo *Store Manager* legge i dati scaricati nelle collezioni, dove una collezione definisce *Kpi* caratterizzati dalle stesse metriche e dimensioni;
5. Lo *Store*, che risiede nel database NoSQL MongoDB, raccoglie il totale delle collezioni;
6. MongoDB si referenzia al *Kpi* tramite il campo mongo_db_store (ove presente) salvato nella tabella *Kpi* del database SQL MySQL.

4.3 DATA LOGIC

Data logic si occupa di gestire le funzionalità di memorizzazione ed accesso ai dati relativi ai modelli Django memorizzati in database relazionale MySQL e a quelli di *Web Analytics* riservati al database NoSQL MongoDB.

4.3.1 MySQL

MySQL è un sistema di database relazionale open source nato nel 1996 per opera dell'azienda svedese Tcx, basato su un DBMS relazionale preesistente chiamato mSQL.
Il progetto è stato distribuito in modalità libera per favorirne la diffusione.
Dal 1996 ad oggi, MySQL si è affermato molto rapidamente prestando le proprie capacità a moltissimi software e siti Internet.

4.3.1.1 Caratteristiche

Tra le caratteristiche di MySQL si evidenziano in particolare:

- alta efficienza nonostante la mole di dati affidati;
- integrazione di tutte le funzionalità che offrono i migliori DBMS, *indici*, *trigger* e *stored procedure* ne sono un esempio;
- altissima capacità di integrazione con i principali linguaggi di programmazione, ambienti di sviluppo e suite di programmi da ufficio.

L'integrazione con il Web framework Django è stata resa estremamente semplice, essendo richiesta la sola configurazione del file di impostazione del progetto, *settings.py*, come segue:

```
DATABASES = { 'default': { 'ENGINE': 'django.db.backends.mysql', 'NAME': 'Nome del database', 'USER': 'Utente del database', 'PASSWORD': 'Password utente', 'HOST': 'Dove il database risiede', 'PORT': '3306', } }
```

In tale modo si è quindi indicato a Django di usare il database di motore MySQL.

La descrizione completa delle tabelle e diagramma UML delle stesse è riportata in Appendice.

4.3.1.2 Criticità

L'aggregazione e la gestione delle tabelle a livello *Utente* in associazione con *Cliente*, *Viste* e dati di ranking *SEO*, non ha portato ad altra difficoltà che non fosse la mera gestione delle chiavi esterne tra le tabelle. In considerazione proprio delle suddette tabelle con tipi di dati omogenei, l'approccio SQL si è quindi dimostrato estremamente efficiente.

Per quel che riguarda i dati di *Web-Analytics* provenienti dai diversi *Kpi*, il sistema di database relazionale di gestione del dato (*schema-full*) si è rivelato fallimentare non potendosi stabilire a priori la variabilità dei tipi di dato per ciascun *Kpi*.
La complessità di gestione, derivante dall'eterogeneità dei vari contesti, necessita di un sistema diverso dalla strutturazione rigida dei contenuti, tipica dei database relazionali. Tale elemento è completamente assente nei database NoSQL.
Per la gestioni di questi dati risulta pertanto fondamentale è un approccio *schema-less*.

Riporto in tabella un esempio di differenti metriche e dimensioni relative a due *Kpi*:

- *"Sessions"*, come *Kpi* che raccoglie i dati delle sessioni complessive degli utenti transitati dalla vista del sito di riferimento;
- *"Click on Promo"*, come *Kpi* che considera le visite degli utenti transitati dalla pagina di promozione.

Confronto tra Kpi

Kpi	Metriche	Dimensioni
Sessions	ga:sessions	ga:date ga:deviceCategory
Click on Promo	ga:goal3Completions ga:goal6Completions	ga:date ga:deviceCategory

4.3.2 HDF5

HDF5 è un formato di file binario appartenente alla famiglia Hierarchical Data Format (HDF), progettato per memorizzare e organizzare grandi quantità di dati.
Originariamente sviluppato presso il National Center for Supercomputing Applications, è supportato dal Gruppo HDF, società senza scopo di lucro la cui missione è quella di garantire la continuità dello sviluppo di tecnologie HDF5.

4.3.2.1 Caratteristiche

La struttura di un file HDF5 [10] si compone di due grandi oggetti:

1. *Dataset*, vettori multidimensionali di un tipo omogeneo;
2. *Gruppi*, strutture di contenitori che possono ospitare Dataset e altri gruppi.

Ciò si traduce in un formato di dati system-like estremamente gerarchico. Infatti, le risorse di un file HDF5 sono anche accessibili tramite sintassi "/" di percorso. HDF5 permette dunque una organizzazione di dati all'interno di file sulla falsa riga di un file-system UNIX, in termini di Gruppi (directory) e Dataset (file).
I dati possono essere descritti da "attributi", sia sui *Gruppi* che sulle directory. I dati archiviati sono binari, ma HDF5 si fa carico di ricordare la piattaforma che li ha generati e di provvedere alla eventuale conversione in modo automatico, essendo "*platform independent*" [11].

Questo formato di file è supportato da molte piattaforme software commerciali e non commerciali, tra cui evidentemente Python.
La distribuzione HDF di tipo open-source è costituita da libreria, utilità della riga di comando, ambiente di test, interfaccia Java e applicazione HDF Viewer basato su Java (HDFView).

4.3.2.2 Criticità

L'approccio gerarchico di HDF5 porta ad una semplificazione nel salvataggio di *Kpi* con metriche e dimensioni identiche, permettendo di aggregare *Kpi* "affini" nello stesso file. Ciononostante la gestione di questo tipo di file ha determinato una serie di criticità.

Le operazioni di lettura e, ove richiesto, di aggiornamento del dato sono difficoltose in quanto:

- l'accesso al contenuto richiede l'uso dell'applicativo HDFView;
- le interrogazioni sui record (ad esempio su base giornaliera) necessitano di essere supportate da codice auto sviluppato.
- l'aggregazione dei record di ogni *Kpi* impone la lettura in memoria dell'intero file, e con l'aumentare delle informazioni memorizzate richiede un costo esoso in termini di risorse.

4.3.3 MongoDB

MongoDB [12] è un database NoSQL orientato ai documenti, basato sul formato BSON per la memorizzazione e la rappresentazione dei dati.

Sviluppato inizialmente dalla società di software 10gen (ora diventata MongoDB Inc.) nell'ottobre 2007 come componente di un prodotto di platform as a service, l'azienda si è spostata verso un modello di sviluppo open source nel 2009.

Da allora, MongoDB è stato adottato come backend da un alto numero di grandi siti Web e società di servizi come Craigslist, eBay, Foursquare, SourceForge e il New York Times, tra gli altri.

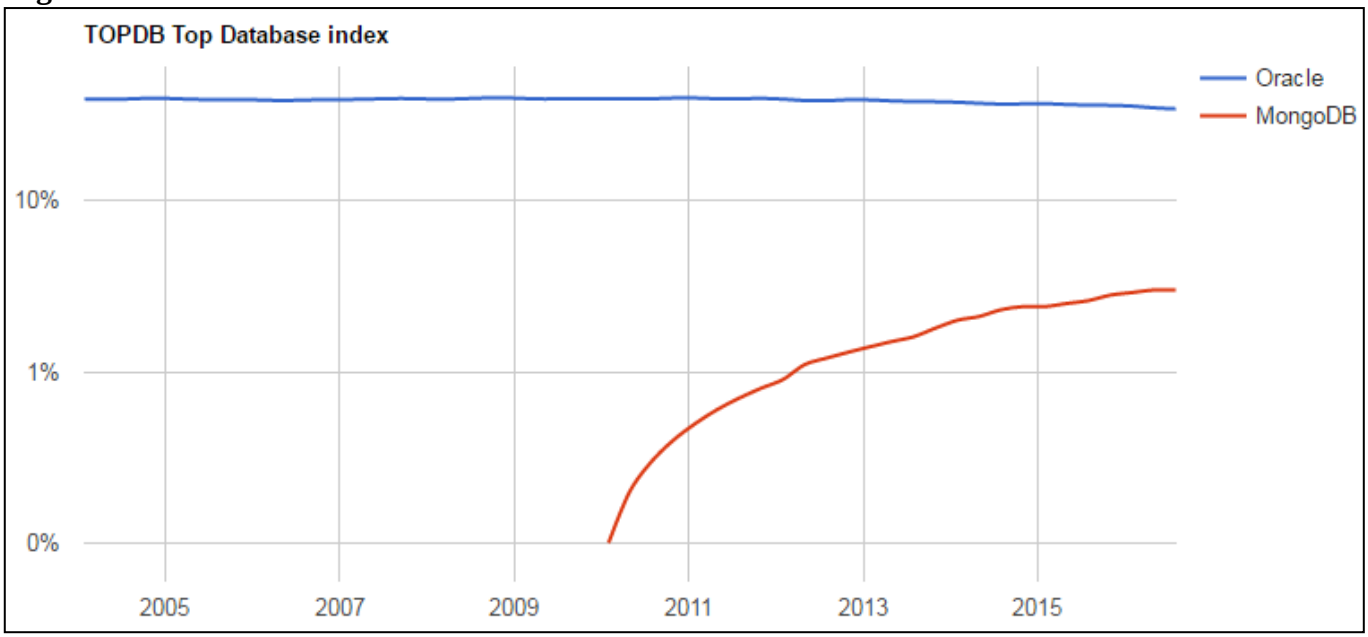
Nel Magic Quadrant 2015 per "Operational Database Management Systems" [13], Gartner ha inserito MongoDB nel quadrante dei "Leader" a livello mondiale (Fig. 14).

Figura 14:

In aggiunta PYPL [14] ha analizzato (Settembre 2016) quanto spesso i sistemi di database sono stati ricercati su Google.

I dati che ne sono scaturiti (Fig. 15) evidenziano come MongoDB rappresenti il primo database NoSQL e sia, complessivamente, quello con il più alto margine di crescita negli ultimi 5 anni (2.3%).

Figura 15:



4.3.3.1 Caratteristiche

Classificato come un database di tipo NoSql, Mongo si allontana dalla struttura tradizionale basata sulle tabelle dei database relazionali in favore di documenti in stile JSON ("JavaScript Object Notation") con schema dinamico (Mongo chiama il formato BSON, ovvero "Binary JSON"), rendendo l'integrazione dei dati di alcuni tipi di applicazioni più facile e veloce.

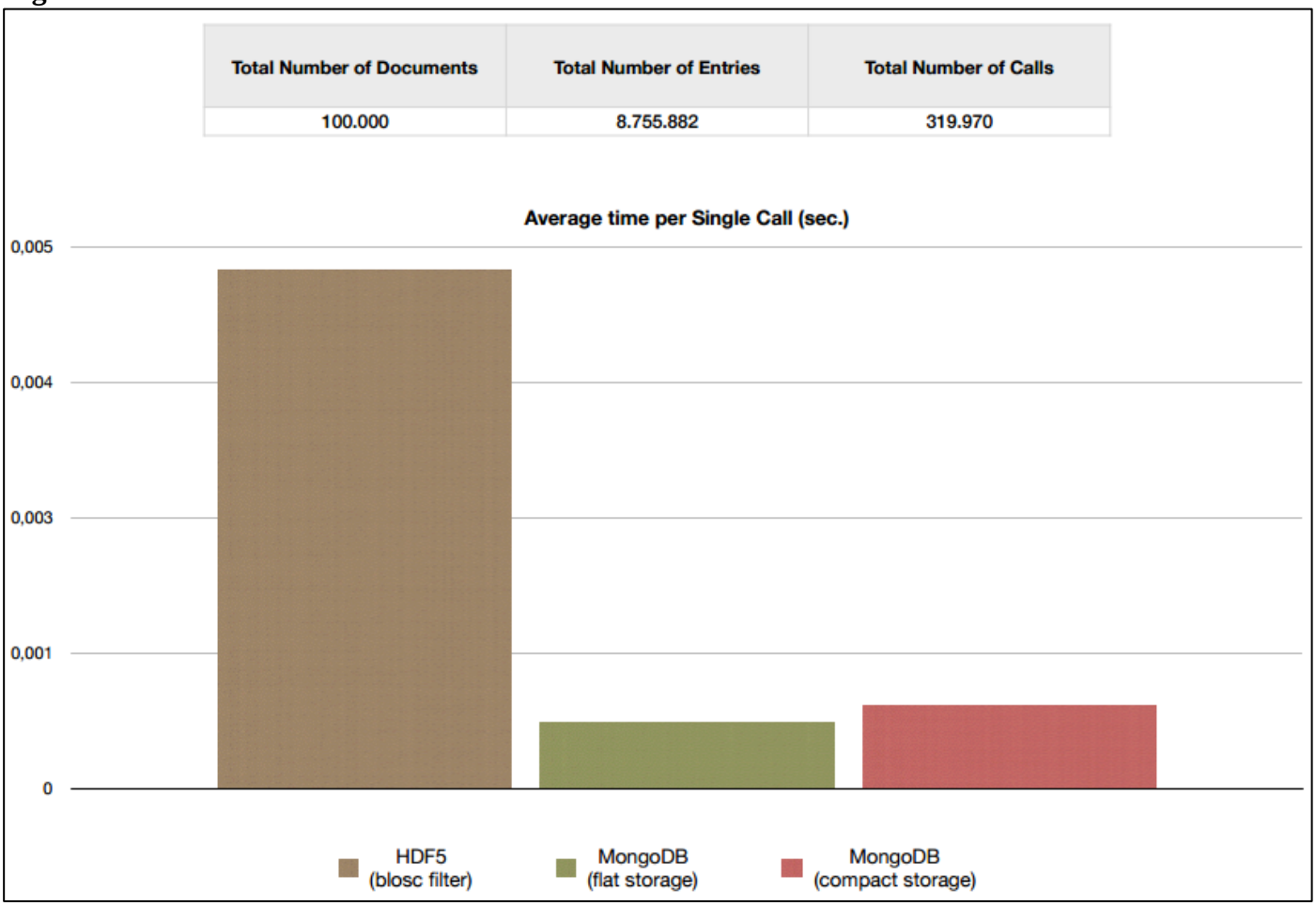
I documenti vengono raccolti in collezioni (collection) che rappresentano una suddivisione concettuale dei dati in modo analogo alle tabelle dei database relazionali; a differenza di queste ultime, in MongoDB una collezione non ha però vincoli strutturali pertanto i documenti presenti all'interno possono avere campi diversi, per tipo e valore.

Le operazioni sul database vengono eseguite in console tramite JavaScript e questo ne consente l'uso immediato ad un gran numero di utenti non contemplando la necessità di imparare particolari sintassi o linguaggi [15].

MongoDB supporta ricerche per campi, intervalli e espressioni regolari. Le interrogazioni possono restituire campi specifici del documento e anche includere funzioni definite dall'utente in JavaScript.

Di seguito riporto le differenze prestazionali tra MongoDB e l'approccio HDF5 (Fig. 16), ottenute da una ricerca presentata da Valerio Maggio alla conferenza europea sul linguaggio Python, l'EuroPython, tenutasi nel Luglio 2016 a Bilbao [16].

Figura 16:



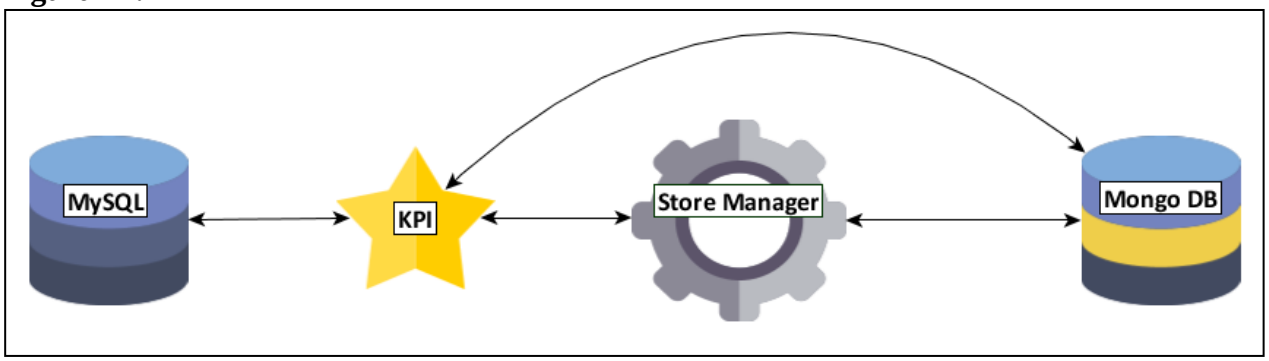
4.3.3.2 Fine Tuning tra MySQL e MongoDB

La gestione dei dati memorizzati in insieme di documenti (collection) MongoDB necessita di essere referenziata al database MySQL (Fig. 17).

Se specificato, il campo "mongodb_store" presente nella tabella *Kpi* è riferito alla collezione del database MongoDB che ne raccoglie quindi i dati scaricati.

Qualora non specificato, l'associazione tra *Kpi* e dato è demandata a Store Manager, classe Python auto sviluppata per l'intera gestione degli stessi.

Figura 17:



In sintesi uno Store Manager è vincolato dalla piattaforma sorgente, Source Platform es. Google Analytics o Autority Labs) che ne definisce i metodi secondo l'approccio CRUD, create read update delete.

4.4 BUSINESS LOGIC

Business logic si occupa di gestire la funzionalità applicativa, ovvero corrisponde a una serie di moduli integrati in un application server per la generazione ed il controllo dei contenuti.

4.4.1 Python

Molti erano i linguaggi di programmazione a disposizione utilizzabili per la realizzazione dell'applicazione.

Dovendo effettuare una scelta, si è puntato ad individuare un linguaggio di programmazione orientato agli oggetti che permettesse lo sviluppo di pagine Web lato server.

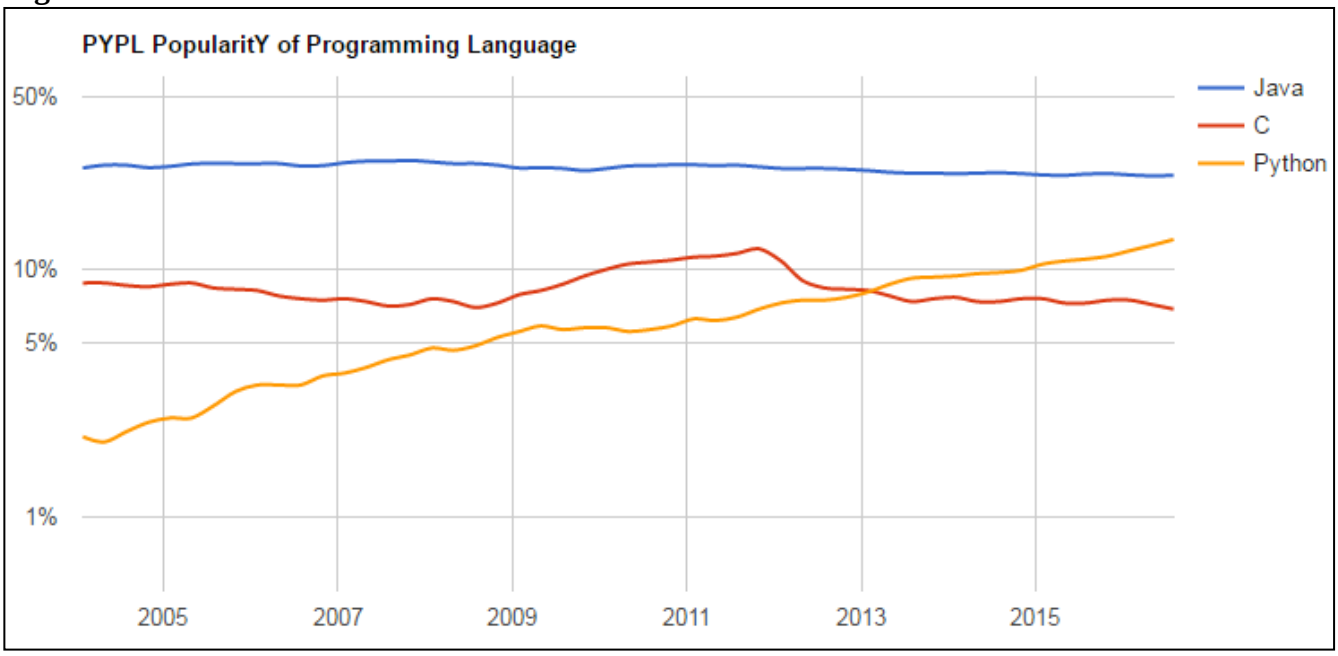
Si è cercato di trovarne dunque uno ben documentato, ed ampiamente utilizzato nel campo di data science e non.

PYPL [17] analizza le ricerche dei tutorial di programmazione su Google, un indicatore rilevante della popolarità di un linguaggio e delle prospettive lavorative che la sua conoscenza può offrire (Fig. 18).

Python [18], una tecnologia in circolazione da circa 25 anni, si piazzerebbe ora al secondo posto con l'11.3% di share, con un +1.1% rispetto all'anno passato ed un notevole +5% negli ultimi cinque anni.

Python avrebbe poi surclassato PHP, componente essenziale dello stack LAMP (Linux, Apache, MySQL, PHP) su cui attualmente poggiano centinaia di milioni di siti Web.

Figura 18:



Uno studio recente (Fig. 19) effettuato da Kdnuggets [19], mostra Python in seconda posizione tra i migliori tool di data science (intesi congiuntamente come linguaggi di programmazione con relative librerie e come prodotti software), evidenziando inoltre una preponderante crescita rispetto all'anno 2015 (+51%).

Figura 19:

Top Analytics/Data Science Tools

Tool	2016 % share	% change	% alone
R	49%	+4.5%	1.4%
Python	45.8%	+51%	0.1%
SQL	35.5%	+15%	0%
Excel	33.6%	+47%	0.2%
RapidMiner	32.6%	+3.5%	11.7%
Hadoop	22.1%	+20%	0%
Spark	21.6%	+91%	0.2%
Tableau	18.5%	+49%	0.2%
KNIME	18.0%	-10%	4.4%
scikit-learn	17.2%	+107%	0%

Sulla base di tali considerazioni la scelta è quindi ricaduta su Python, un linguaggio estremamente leggibile e portabile con supporto a diversi paradigmi di programmazione, fornito di una estesa libreria standard e di moltissima documentazione ufficiale e non.
Il linguaggio Python nasce ad Amsterdam nel 1989, dove il suo creatore Guido Van Rossum lavorava come ricercatore.

4.4.1.1 Caratteristiche

Python è un linguaggio multi-paradigma. Permette infatti di scrivere programmi agevolmente seguendo molteplici paradigmi, quali object oriented, programmazione strutturata o funzionale.
Il controllo dei tipi è forte (strong typing) e viene eseguito runtime (duck typing). In altre parole una variabile può assumere nella sua storia valori di tipo diverso, pur appartenendo in ogni istante ad un tipo ben definito.
Python utilizza un garbage collector per la gestione automatica della memoria.
Python possiede un gran numero di tipi base. Oltre ai tipi interi e floating point classici, supporta in modo trasparente numeri interi arbitrariamente grandi e numeri complessi, nonché tutte le operazioni classiche sulle stringhe con una sola eccezione: le stringhe in Python sono oggetti immutabili, cosicché qualsiasi operazione che vada in qualche modo ad alterare una stringa (come ad esempio la semplice sostituzione di una parte di essa) restituirà una nuova stringa.
Altro punto di forza in Python è la disponibilità di elementi che facilitano la programmazione funzionale. Python permette infatti di avere funzioni come argomenti.
Se paragonato ai linguaggi compilati tipizzati in modo statico, come ad esempio il C, la velocità di esecuzione non è uno dei punti di forza di Python, specie nel calcolo matematico.
Tuttavia, Python permette di aggirare in modo facile l'ostacolo delle performance pure, risultando relativamente semplice scrivere un'estensione in C o C++ da utilizzarsi all'interno di Python, in modo da sfruttare l'elevata velocità di un linguaggio compilato solo nelle parti in cui effettivamente necessario, e sfruttando invece la potenza e la versatilità di Python per tutto il resto del software.
Python ha una vasta libreria standard, il che lo rende adatto a molti impieghi. Oltre ai moduli della libreria standard se ne possono poi aggiungere altri scritti in C oppure in Python per soddisfare le proprie esigenze particolari. Ad eccezione di poche funzioni la libreria standard è completamente compatibile con tutte le piattaforme [20].

4.4.1.2 Pip e Virtualenv

Per gestire le dipendenze del server Python del progetto è stato usato Pip [21], il cui acronimo ricorsivo sta per Pip Installs Packages. Si tratta dello standard di fatto per la gestione delle dipendenze dei programmi Python.
Lo standard di Pip prevede infatti di installare le librerie globalmente, non consentendo quindi di creare un ambiente locale dove mantenere le dipendenze.
Un'alternativa per la gestione pulita delle dipendenze locali di progetto è Virtualenv [22], un comando per creare un ambiente Python locale.
Virtualenv è stato quindi utilizzato per creare un ambiente specifico per il progetto, gestendo così localmente tutte le librerie necessarie alla corretta esecuzione dell'applicazione.

4.4.1.3 Librerie per Computing Scientifico

Python ha una posizione forte nell'ambito del Computing Scientifico:

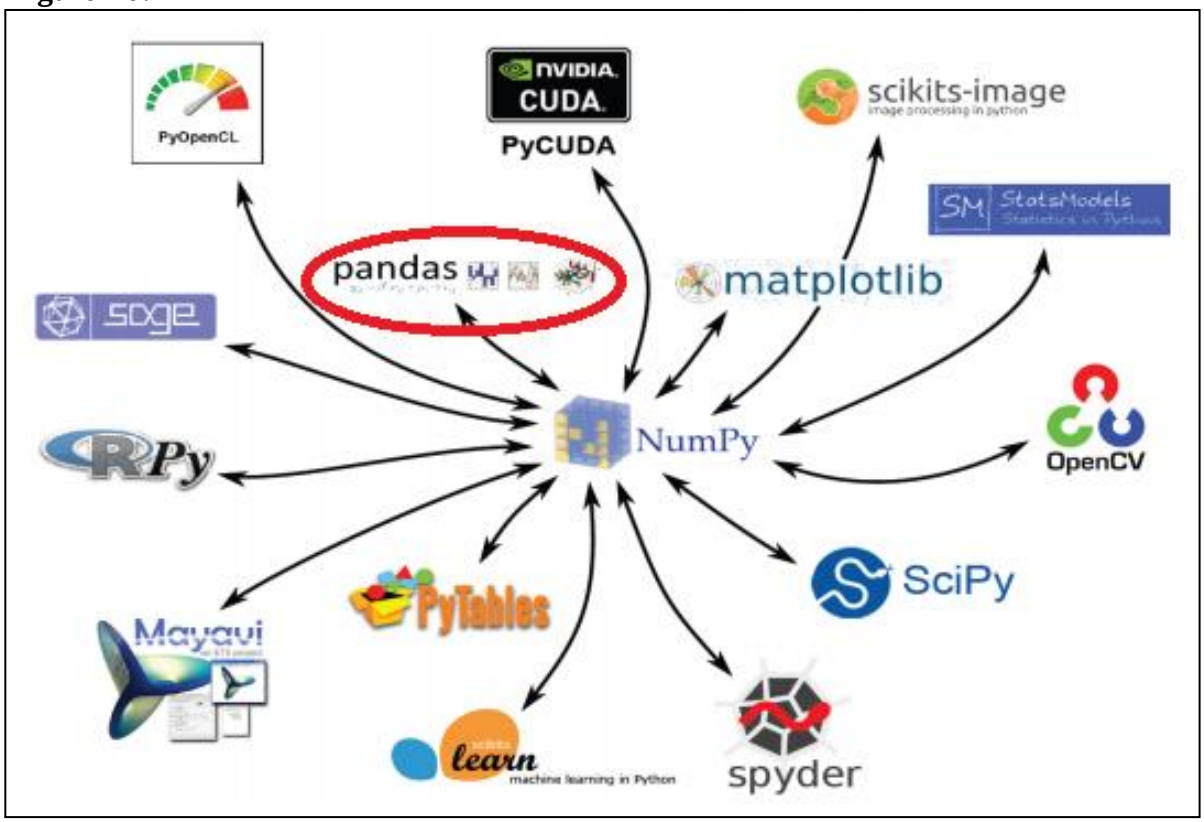
- è open-source;
- le comunità di utenti sono decisamente significative e diffuse, risulta quindi piuttosto facile reperire in rete aiuto e documentazione;
- ad oggi è disponibile un esteso ecosistema di librerie scientifiche;
- si possono scrivere codici Python che ottengono ottime performance, grazie alla forte integrazione con prodotti altamente ottimizzati scritti in C e Fortran.

NumPy [23] è il pacchetto base per il calcolo scientifico con Python. Tra le caratteristiche principali:

- gestione degli array N-dimensionali;
- integrazione con C, C++ e Fortran;
- funzioni di base di algebra lineare.

NumPy rappresenta il cuore (Fig. 20) della maggior parte delle librerie scientifiche di Python [24].
In particolare, per la aggregazione e la lettura dei dati, si è fatto riferimento a Pandas [25].
La libreria, inizialmente sviluppata nel 2008 da Wes McKinney presso una banca d'affari per semplificare la lettura e la gestione dei dati finanziari, ha riscosso un immediato successo nella comunità scientifica Python e molti sviluppatori hanno contribuito all'evoluzione della stessa. Questo solo per sottolineare la flessibilità e versatilità di questo linguaggio di programmazione.

Figura 20:



Pandas fornisce strutture dati di alto livello e strumenti ottimizzati per la manipolazione e la generazione di dati di tipo relazionale e tabellare.

Le tipologie di dato per cui la libreria è ottimizzata sono le seguenti:

- *Series*, serie (di tipo monodimensionale) temporali e non con frequenza variabile;
- *DataFrame*, dati di tipo bidimensionale in forma tabellare con colonne di tipi eterogenei.

Un *DataFrame* può essere costruito passando un dizionario di liste di uguale lunghezza. Questa è la tipica situazione sfruttata nella aggregazione dei dati dei *Kpi* a seguito della lettura della rispettive collezioni MongoDB.

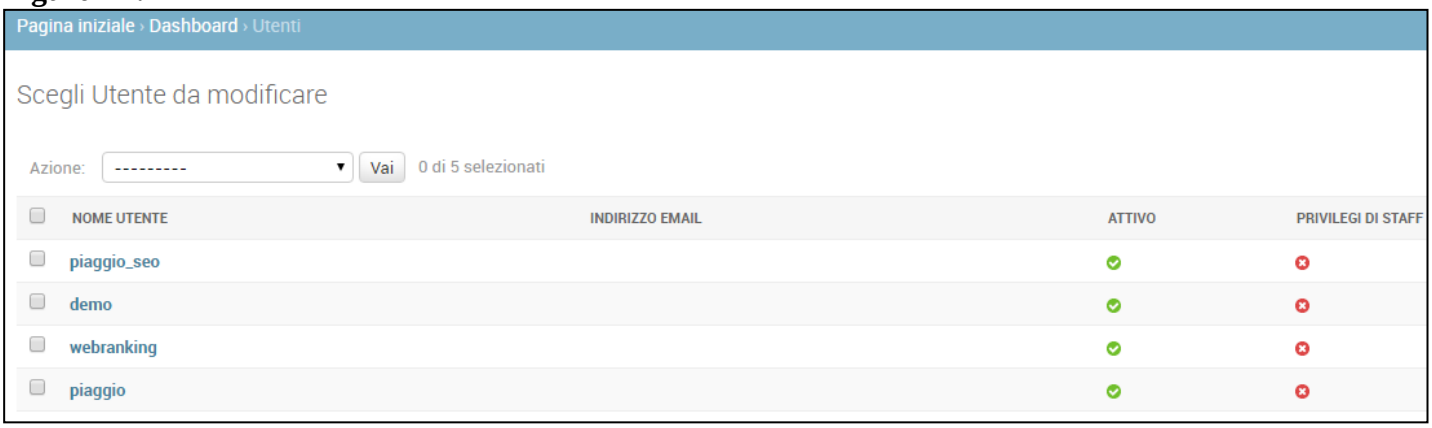
Riporto ora un semplice esempio di creazione e stampa di un *DataFrame* a partire da un dizionario di *Series*:

```
In [32]: d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']),
   ....:      'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
   ....:

In [33]: df = pd.DataFrame(d)

In [34]: df
Out[34]:
   one  two
a  1.0  1.0
b  2.0  2.0
c  3.0  3.0
d  NaN  4.0
```

4.4.2 Django

Django [26] è un Web application framework open-source scritto in Python rilasciato a Luglio 2005 per gestire diversi siti di notizie per conto della World Company di Lawrence del Kansas.
Tale framework è diventato un must nello sviluppo Web, basti pensare che tra i siti Web che lo utilizzano vi sono attualmente Pinterest, Instagram, Mozilla, The Washington Times.
Django offre una serie di funzionalità che facilitano lo sviluppo rapido di applicazioni per la gestione di contenuti, enfatizzando la riusabilità dei componenti secondo il principio *DRY* (*Don't Repeat Yourself*).
Sono forniti inoltre moduli opzionali di amministrazione generati automaticamente che consentono di leggere, creare, aggiornare ed eliminare contenuti rappresentati come oggetti e di tenere traccia di tutte le operazioni effettuate.
L'interfaccia grafica rende ad esempio possibile gestire utenti e gruppi di utenti, inclusi i permessi a loro associati (Fig. 21).

Figura 21:



Il cuore del framework Django è basato sul paradigma *Model View Controller* (*MVC*) che fa da intermediario tra i modelli dei dati del database relazionale (*Model*), il sistema per la gestione di template (*View*) e l'instradatore delle richieste applicative tramite URL (*Controller*).

4.4.2.1 Modelli Django

La creazione delle tabelle del database MySQL è demandata ai modelli Django. Un modello in Django è quindi uno speciale tipo di oggetto salvato nel database.

Le proprietà di ciascun modello sono definite dai suoi tipi come segue, per citarne alcuni:

- models.CharField, come testo con un numero limitato di lettere;
- models.TextField, come testo senza un limite;

- models.DateField, come campo per la data;
- models.ForeignKey, come chiave esterna riferita ad un altro modello;
- JSONField, come campo per la memorizzazione di dati in formato JSON validato.

Le proprietà di ciascun campo sono legate ai suoi descrittori.

In poche parole, un descrittore Python è un oggetto che rappresenta il valore di un attributo. Di conseguenza, se un oggetto ha un nome di attributo, un descrittore è un altro oggetto che può essere usato per rappresentare il valore contenuto in quell'attributo, nome.

Un esempio di descrittore definisce una serie di metodi (*get, set, delete*) [27] che vincolano l'oggetto a un tipo di dato:

```
class TypedProperty(object):
    def __init__(self, name, type, default=None):
        self.name = "_" + name
        self.type = type
        self.default = default if default else type()
    def __get__(self, instance, cls):
        return getattr(instance, self.name, self.default)
    def __set__(self,instance,value):
        if not isinstance(value,self.type):
            raise TypeError("Deve essere %s" % self.type)
        setattr(instance,self.name,value)
    def __delete__(self,instance):
        raise AttributeError("Non posso cancellare l'attributo")

class Foo(object):
    nome = TypedProperty("nome", str)
    num = TypedProperty("num", int, 42)

>> acct = Foo()
>> acct.nome = "obi"
>> acct.num = 1234
>> print acct.num → '1234'
>> print acct.nome → 'obi'
>> acct.num = '1234' → TypeError: Must be a <type 'int'>
```

Si può notare come l'assegnazione di una stringa al campo num, quindi definito da tipo numero intero, produca un errore.

4.4.3 RESTFUL API

L'approccio tipico di RESTFUL API consiste nella creazione e l'utilizzo di procedure per l'applicazione di un determinato compito nella comunicazione di tipo client-server.
L'interfaccia di programmazione di una applicazione (API) contribuisce ad ottenere un astrazione di alto livello, semplificando nello specifico la comunicazione tra *presentation logic* e *data logic*.
Il paradigma REST (*REpresentional State Trasfer*) impone per definizione una tipologia architetturale client-server orientata alla risorsa (*Resource Oriented Architecture*), ove per risorsa si intende un'entità a sé stante identificata in maniera descrittiva da URI (*Uniform Resource Identifier*).

Le proprietà fondamentali di questo paradigma sono:

- *addressability*, come esposizione univoca del dato mediante l'uso delle risorse, in particolare garantisce di specializzare il livello di dettaglio della risorsa mediante semplici parametri;
- *statelessness*, come capacità di risolvere ad una ad una le richieste per poi deallocare le risorse;
- *connectedness*, come possibilità di avere collegamenti esterni alle risorse;
- *uniform interface*, come interfaccia uniforme con metodi standard (POST, GET, PUT, DELETE);
- *cachable*, come possibilità di memorizzare le richieste risolte in cache.

4.4.3.1 Django REST Framework

Django REST Framework [28] è un plugin aggiuntivo per la creazione di interfacce REST.

Tra le funzionalità incluse in Django REST Framework spiccano le seguenti:

- integrazione con sistemi di autenticazione di tipo OAuth2;
- Web browseable API, come funzionalità che permette di navigare l'interfaccia REST di risposta alle richieste API direttamente da browser;
- serializzazione degli oggetti inviati in risposta secondo formati diversi (XML, JSON, e la stessa Web browseable API serializzata come HTML, …);
- estensione delle viste (*View*) di Django allo sviluppo di interfacce REST tramite modello, serializzatore ed azione da applicare (basata su principio *CRUD*).

Riporto di seguito un esempio di risposta JSON alla richiesta delle categorie di *Kpi* attive per il cliente selezionato:

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "nome": "Traffic",
        "fontawesome_icon": "expand",
        "order": 1
    },
    {
        "id": 2,
        "nome": "Time Spent on Site",
        "fontawesome_icon": "clock-o",
        "order": 2
    },
    {
        "id": 3,
        "nome": "Action",
        "fontawesome_icon": "credit-card",
        "order": 3
    },
    {
        "id": 4,
        "nome": "Seo",
        "fontawesome_icon": "search-plus",
        "order": 4
    }
]
```

4.5 PRESENTATION LOGIC

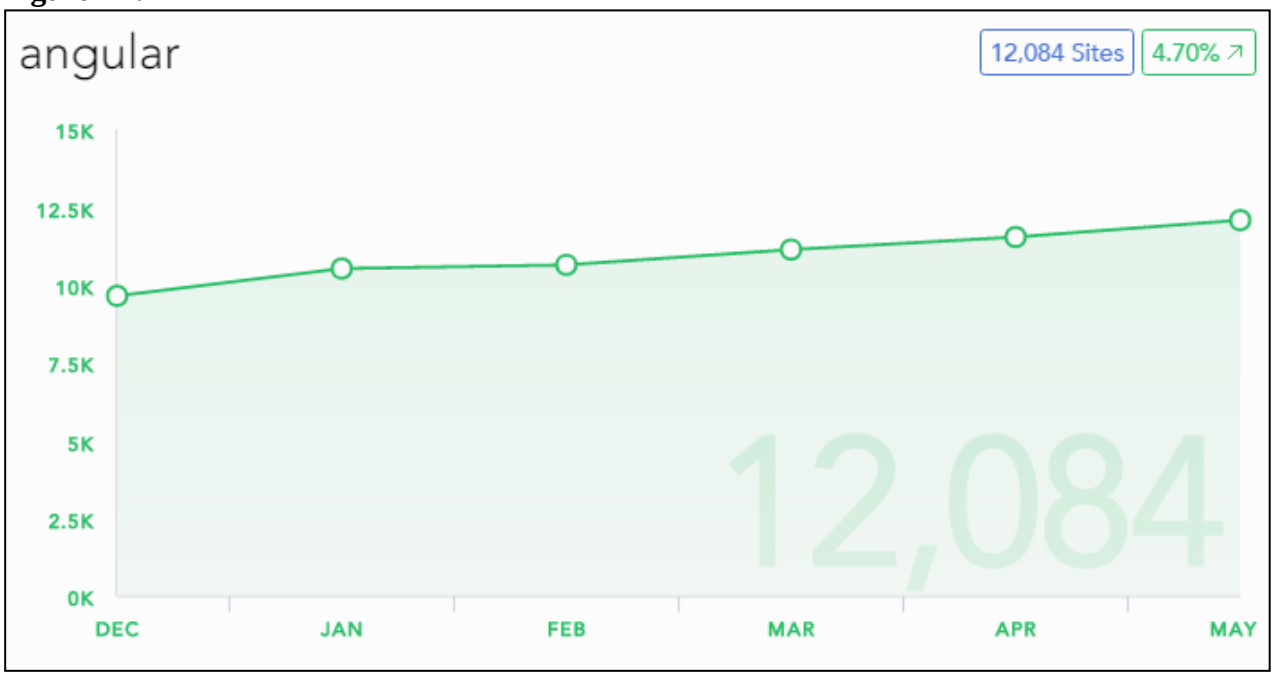
Presentation Logic costituisce l'interfaccia che si pone tra la user interface (ciò che l'utente percepisce interagendo con il servizio) e la business logic, rappresentando pertanto ciò che accade quando l'utente interagisce con l'interfaccia dell'applicazione.

4.5.1 AngularJS

AngularJS [29] è un framework open-source volto alla semplificazione nella realizzazione di *Single Page Applications* (*SPAs*) sviluppato nel 2009 da Miško Hevery, cui è valsa l'assunzione a Google. Secondo Libscore [30], tra Dicembre 2015 e Maggio 2016 il framework è cresciuto del

4.70%, mentre dei 12.084 siti che attualmente ne fanno uso si ricordano NBC, Walgreens, Intel, Sprint e ABC News (Fig. 22).

Figura 22:



Si tratta di un framework totalmente compatibile con la maggior parte delle librerie Javascript. AngularJS gode di alcune proprietà distintive quali:

- *dependency-injection*, come gestione delle dipendenze tra i vari moduli;
- *direttive HTML5 ready*, come facoltà di utilizzare solo il linguaggio HTML5 per abilitare le funzioni dei controller e delle altre componenti;
- logica dichiarativa con approccio *DRY* per la semplificazione del riuso dei componenti che consente di diminuire drasticamente il codice necessario.

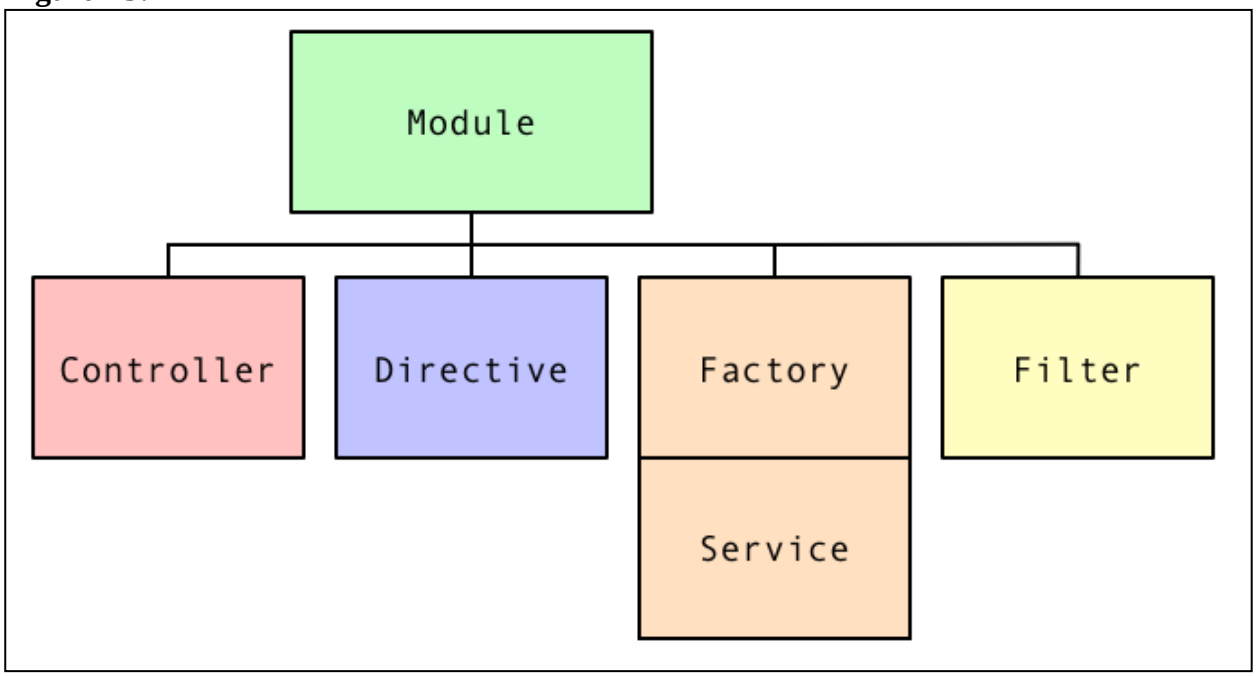
Il framework si basa sul paradigma architetturale *Model View Whatever* (*MVW*) e le principali componenti che lo caratterizzano sono:

- *directive*, come possibilità di estendere i tag HTML5 definendone di nuovi, indipendenti dal resto dell'applicazione e riutilizzabili in altri progetti in maniera dichiarativa;
- data-binding bidirezionale affidato al *controller*, come meccanismo per sincronizzare viste e modelli (mediante *$scope*), in modo tale che ogni cambiamento di ciascun modello venga immediatamente riflesso nella componente della vista e viceversa;
- *module*, come metodo di inizializzazione della applicazione consentendo facile incapsulamento di librerie;
- *service* e *factory*, come incapsulamento di classi comuni in altre componenti;

- *filter*, come capacità di filtrare e modificare gli oggetti del modello sulla base di specifiche definite dal programmatore.

Di seguito è riportato lo schema funzionale relativo alle macrocomponenti sopra citate (Fig. 23).

Figura 23:



Il lato front-end della *VAD* necessita di gestire e visualizzare un elevato numero di dati sotto diverse specifiche di formato (grafico, tabella, ecc.). Proprio per adempiere a tali esigenze è stato deciso di utilizzare AngularJS, che mette a disposizione numerose funzioni per generare elementi in pagina in modo dinamico.

4.5.2 Librerie JavaScript

La complessa logica di presentazione della applicazione ha reso necessario l'utilizzo di numerose librerie JavaScript:

- *jQuery* [31], per facilitare la gestione dei documenti HTML e i relativi elementi mediante sintassi semplificata, oltre ad essere rivolta alla coordinazione degli eventi;
- *Restangular* [32], libreria AngularJS volta a semplificare le richieste di tipo CRUD dirette alle RESTFUL API;
- *D3.js* [33], per la creazione di grafici dinamici a partire dai dati (D3 è l'acronimo *di Data-Driven Documents*);
- *NVD3* [34], libreria AngularJS per l'utilizzo dei grafici D3.js reimpostati;
- *Lodash* [35], per la manipolazione del dato, grazie ad oltre un centinaio di funzioni raggruppate sotto diverse categorie definite sulla base del tipo di dato considerato.

4.6 DEPLOY DELLA APPLICAZIONE SU AMAZON

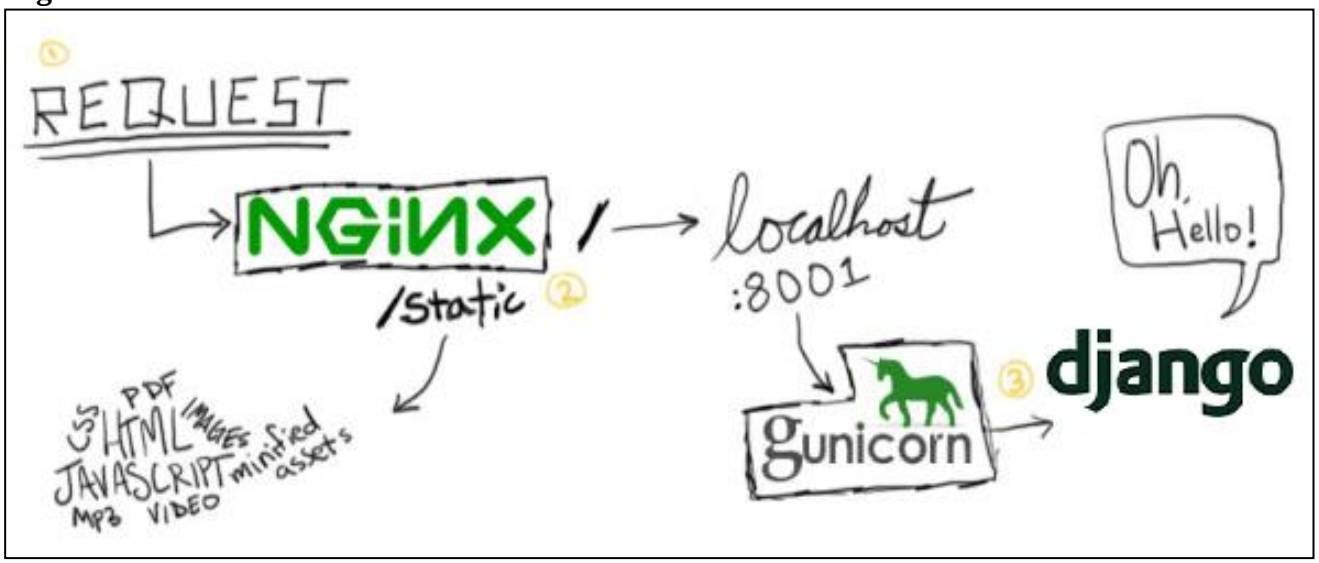
Il paragrafo presenta gli strumenti adottati per le operazioni di rilascio dell'applicazione *VAD* su Amazon Web Services.

4.6.1 DUNG Stack

Per ospitare una applicazione scritta con il framework Django si inizialmente è valutato se adottare lo stack in stile LAMP (Linux + Apache + mod_wsgi).
Successivamente per cercare di limitare l'utilizzo di Ram e CPU, grazie anche ad un articolo proveniente dal blog degli ingegneri di Instagram [36], è stato invece scelto lo stack DUNG che rappresenta la configurazione con Django + Ubuntu + Nginx + Gunicorn. La configurazione di Webserver e application server si è dimostrata peraltro più semplice seguendo questo stack.

Il rilascio (*deploy*) basato DUNG è composto da (Fig. 24):

Figura 24:



- Django come Web framework;
- Ubuntu server versione 11.04;
- Nginx [37] come Web server leggero ad alte prestazioni, che si occupa di servire rapidamente contenuti statici. Nginx utilizza un approccio asincrono basato su eventi nella gestione delle richieste in modo da ottenere prestazioni maggiormente prevedibili sotto stress, in contrasto con il modello del server HTTP Apache che usa un approccio orientato ai thread o ai processi nella gestione delle richieste;
- Gunicorn [38] come application server WSGI. In questo caso è stato necessario configurare Nginx come proxy server per la risoluzione delle richieste. In tale modo

ciascuna richiesta inerente il codice python deve essere trasmessa al server WSGI (Gunicorn) che eseguirà e restituirà la pagina risultante.

4.6.2 Fabric

Fabric [39] è un libreria Python e uno strumento a linea di comando per semplificare le operazioni di deploy tramite connessioni SSH.

Fornisce una suite di operazioni per eseguire comandi locali o remoti, di norma con diritti di superutente (*sudo*).

Questo sistema ha reso possibile l'esecuzione di comandi su più istanze alla volta, come aggiornamento del codice nel server in produzione e migrazione delle tabelle, scaricamento del database relazionale MySQL (riportato in Appendice) o, analogamente, importazione dei dati Analytics provenienti MongoDB in locale.

4.6.3 Supervisor

Supervisor [40] è un sistema client/server per il controllo dei processi Unix.

Questi sistemi hanno la funzione di garantire che altri processi siano costantemente attivi. Possono essere quindi elementi importanti per assicurare l'affidabilità di un servizio soprattutto nei casi in cui il demone da controllare è stato appena sviluppato e di conseguenza non sempre possiede un alto grado di affidabilità.

Inizialmente è stato preso in considerazione Upstart, il controllore di processi presente nativamente su Ubuntu. Tuttavia questo sistema ha lo svantaggio di perdere il riferimento con ogni processo terminato, poiché la funzione *respawn* prevede che ad ogni processo terminato ne sia lanciato uno nuovo.

Supervisor al contrario traccia lo stato di tutti i demoni registrati, in aggiunta ai comandi di *start*, *stop* e *restart*.

5 TEST E RISULTATI

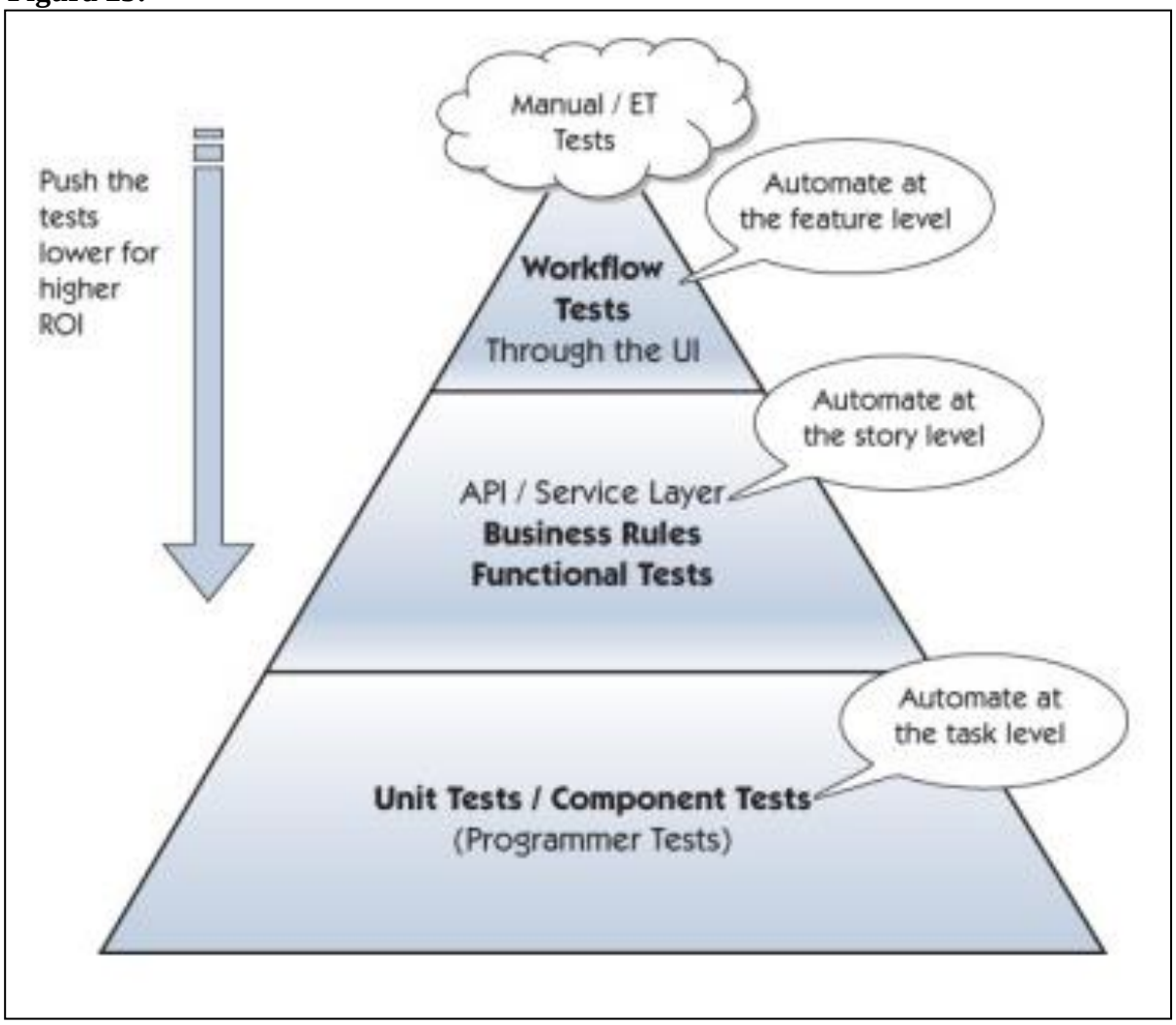
La sezione presenta in primis alcune metodologie di lavoro orientate ai test, mentre il finale è dedicato all'esposizione delle tecniche di ottimizzazione delle performance utilizzate, nonché dei risultati funzionali ottenuti.

Lo sviluppo dei test ha preso forma principalmente dalla teoria denominata *Test Automation Pyramid* di Mike Cohn [41] che, tra l'altro, è uno degli ideatori del metodo *Scrum* di cui si parlerà in seguito (paragrafo 5.1.1).

La piramide (Fig. 25) è divisa in tre macrofasce così ripartite:

- Livello di test unitari relativi all'approccio di sviluppo guidato dai test (*programmer test*);
- Livello di test relativi alle performance (*api/service layer*);
- Livello di test di riguardanti l'interfaccia grafica della applicazione (*through the user interface*).

Figura 25:



In sintesi, visualizzando il disegno precedente si nota che la piramide richiede un approccio di programmazione più "complesso" alla sua base che si semplifica invece procedendo in direzione del vertice ma, nel contempo, l'effettuazione di test di basso livello porta ad un alto ritorno di investimenti (*ROI*).

Aggiungerei che la realizzazione di test di alto livello attraverso l'interfaccia grafica, seppur più facili dal punto di vista tecnico, necessita di un maggiore sforzo sia in termini di tempo che di risorse economiche.

5.1 METODOLOGIE AGILI DI SVILUPPO DEL SOFTWARE

Prima di entrare nel merito delle tecniche applicate nello sviluppo del software, introduco brevemente il concetto di metodologie agili.

L'adozione della filosofia *Agile* [42] permette il rilascio del software in modo iterativo e incrementale al fine di rendere il team di sviluppo in grado di ridurre sensibilmente il rischio di non rispetto dei tempi e/o di commettere errori di interpretazione dei requisiti mediante l'organizzazione del lavoro in piccole iterazioni, chiamate *sprint*, della durata di poche settimane (tipicamente 2-3).

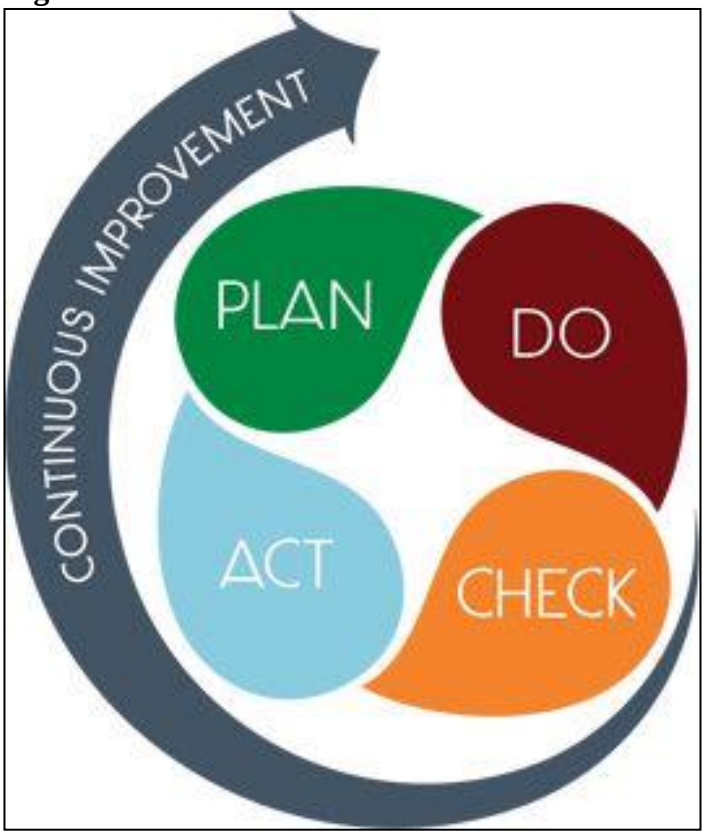
In sintesi un progetto software di entità medio grande può essere suddiviso in tanti progetti di dimensioni più piccole, aventi cicli di lavoro nell'ambito di un singolo *sprint*.

Agile mette quindi in discussione i principi dell'approccio tradizionale del modello a cascata (*waterfall*), facendo in modo che tutte le attività vengano svolte in parallelo durante l'intero ciclo di vita del progetto.

Uno dei requisiti chiave per il buon funzionamento dell'impiego delle metodologie agili è la continua interazione tra sviluppatori e committente al fine di poter condividere, passo dopo passo, una visione chiara ed aggiornata. Esigenze, requisiti e feedback del lavoro svolto da una parte, e stato dell'arte del prodotto dall'altra, nonché la facoltà di rimediare alle cosiddette *smart failure* (fallimenti intelligenti).

Tale processo evolutivo può essere riassunto dal ciclo del modello di *Deming* (*plan, do, check, act*) [43], che contribuisce al processo di miglioramento continuo nello sviluppo di un progetto (Fig. 26).

Figura 26:



5.1.1 Test Driven Development a processo iterativo

Le metodologie agili adottate nello sviluppo della *VAD*:

- Scrum [44], per le attività di gestione del processo iterativo nel suo insieme;
- Extreme Programming (XP) [45], per le attività di Test Driven Development.

Lo Scrum è una metodologia agile iterativa ed incrementale per la gestione dello sviluppo di prodotti software.
Presentata nel 1995 da Jeff Sutherland e Ken Schwaber come metodologia di sviluppo software, è stata adottata da diverse multinazionali (Amazon, Google, Apple, Ferrari, Toyota, etc.) per la gestione dei processi in altre aree aziendali.
Il termine Scrum è mutuato dalla parola inglese che nel gergo del rugby significa “mischia”. In quanto metodologia Agile incoraggia la auto-organizzazione dei team e la comunicazione continua tra membri del team.
Un principio chiave dello Scrum degno di riconoscimento risiede nel fatto che durante lo sviluppo di un progetto il committente può cambiare idea su requisiti e necessità, mutamenti che non sarebbero attuabili in modo agevole e rapido su un progetto gestito con approccio tradizionale.

Scrum pone l'accento sull'abilità del team ad un rilascio rapido passo dopo passo tramite *sprint*, e a una risposta altrettanto veloce ai cambiamenti che emergono, eventualmente anche a discapito della totale comprensione e/o definizione del problema nella fase iniziale, che può quindi essere soltanto parziale.

Un team Scrum è composto da:

- *Product Owner*, come cliente committente;
- *Developer*, come sviluppatore;
- *Scrum Master*, come gestore del lavoro del team, collaborando con il Product Owner alla definizione del progetto affinché i requisiti risultino chiari.

Nel processo di sviluppo della *VAD* si è fatto uso di Teamwork [46], un strumento accessibile via applicazione Web per la gestione dei processi aziendali (*project management tool*).
Fondato nel 2007, è attualmente utilizzato da aziende come PayPal, Ebay, Disney, Spotify, Pepsi, etc.
Tra le caratteristiche principali di Teamwork si sottolinea l'organizzazione del lavoro basata su task, intesi come veri e propri *sprint* simil-Scrum, a cui è possibile assegnare etichette, priorità, scadenze.
In aggiunta, per migliorare la comunicazione intra-team è stato utilizzato Slack [47], piattaforma di messaggistica integrata con una rete di plugin supplementari.
Nel caso specifico di Webranking si è fatto anche uso dell'integrazione con Bitbucket, per tenere traccia delle ultime modifiche (*commit*) apportate alla *VAD*.
Bitbucket di Atlassian [48] è un servizio di hosting web-based per la gestione di repository Git, sistema di controllo di versione volto all'integrazione continua della fase di implementazione del codice.
In particolare si è fatto ampio utilizzo dei cosiddetti branch [49] per l'implementazione ed il test delle nuove funzionalità man mano integrate nel codice radice (master).
Branch, in italiano ramo, consiste in una vera e propria diramazione dalla radice sviluppata in modo indipendente, volta l'implementazione di funzionalità tra loro isolate.
Una volta completato il fine primario del branch, si può applicare il comando di "*merge*" per fonderlo con il master.

L'Extreme Programming (XP) è una metodologia di sviluppo agile che enfatizza il lavoro di squadra, affinché la realizzazione del progetto sia attuata nel modo più efficiente possibile.
Ideata da Kent Beck nel 1997, focalizza lo sviluppo del software sull'approccio definito come Test Driven Development (TDD) che prevede appunto che il codice venga scritto a partire dai test effettuati.

La tecnica TDD consiste nella stesura del codice a partire dalle classi di test che ne assicurano il buon funzionamento.
Spesso la fase di testing, così come la necessità di dover scrivere delle classi di test, è vista come un fastidio da parte degli sviluppatori software e la realizzazione di test nella programmazione della *VAD* non ha fatto eccezione.
Per contro tale potente approccio, garantendo copertura di test (*test coverage*) del codice *bug-sensitive*, ha permesso di ridurre significativamente il rischio di fallimenti inaspettati.
Una volta individuati uno o più scenari *bug-sensitive* riguardanti il comportamento dell'applicazione, gli stessi vengono convertiti in test unitari ove per test unitario si intende appunto l'attività di testing inerente singole unità software.
Utile in questi casi è il paradigma "*Given-When-Then*".
Ad esempio: "dato che (*Given*) 2 diviso 2 risulta 1, quando (*When*) è chiamato il metodo dividi, allora (*Then*) il metodo restituisce 1".

Tra i casi di studio inerenti i test si ricorda:

- l'associazione tra *Kpi* e collezioni MongoDB demandata allo *Store Manager* per *Kpi* affini di cui non sia specificato il campo "mongodb_store" → il test ha verificato che solo *Kpi* con metriche e dimensioni identiche venissero assegnati alla stessa collezione MongoDB;
- il recupero dei dati da API Google Analytics non campionato → il test ha verificato la perfetta corrispondenza con il valore aggregato osservabile da piattaforma Web Google Analytics.

Il processo di analisi dei test effettuato si compone di quattro fasi:

1. scrittura del codice relativo al test guidato da librerie quali "mock" e "django.test", volte al mascheramento dei metodi di classe originari a favore dei test, oltre al metodo "asser_frame_equal" per il confronto di due tabelle di dati *pandas.Dataframe*;
2. lancio e fallimento del test;
3. scrittura del codice per ottenere esisto positivo ai test già scritti (test del punto 2 compreso);
4. miglioramento di caratteristiche non funzionali del codice scritto (*refactoring*) al fine di eliminare i cosiddetti *code smell*, ovvero porzioni di codice che potrebbero, col tempo, portarlo a degenerare come accade nel caso di metodi ripetuti e complessi o del ricorso a variabili inutili, per fare un esempio.

Beneficio chiave dato da questo tipo di approccio procedurale risiede nel fatto che il codice scritto in modo incrementale e soggetto *refactoring* iterativi si conserva semplice e mantenibile.
Come precedentemente accennato, spesso i test vengono accantonati perché ritenuti alla stregua di una perdita di tempo per giunta poco stimolante. Tuttavia la tecnica TDD definisce la scrittura del codice relativa ai test come parte integrante del processo di sviluppo, a scapito di un investimento in termini di tempo che si rivela relativamente contenuto.

5.2 STRESS TEST

La peculiare caratteristica della *VAD* di raccogliere dati da un elevato numero di siti (oltre 200) determina l'insorgere di un problema di scalabilità della piattaforma, con il crescere dell'arco temporale di scaricamento del dato.
Al fine di far fronte al problema, sono state osservate le risorse utilizzando New Relic [50], strumento di monitoring volto a comprendere le performance della applicazione su cui è installato.
Tale strumento già utilizzato da Webranking, è stato installato attraverso il plugin Amazon EC2 che raccoglie dati sul portale New Relic dedicato all'applicazione.
Tali dati rappresentati anche in forma grafica (Figg. 27-28), raccolgono statistiche inerenti:

- operazioni su disco;
- utilizzo di CPU;
- utilizzo di memoria fisica;
- traffico di rete input/output;
- utilizzo di CPU e memoria per ciascun servizio attivo sulla macchina.

Figura 27:

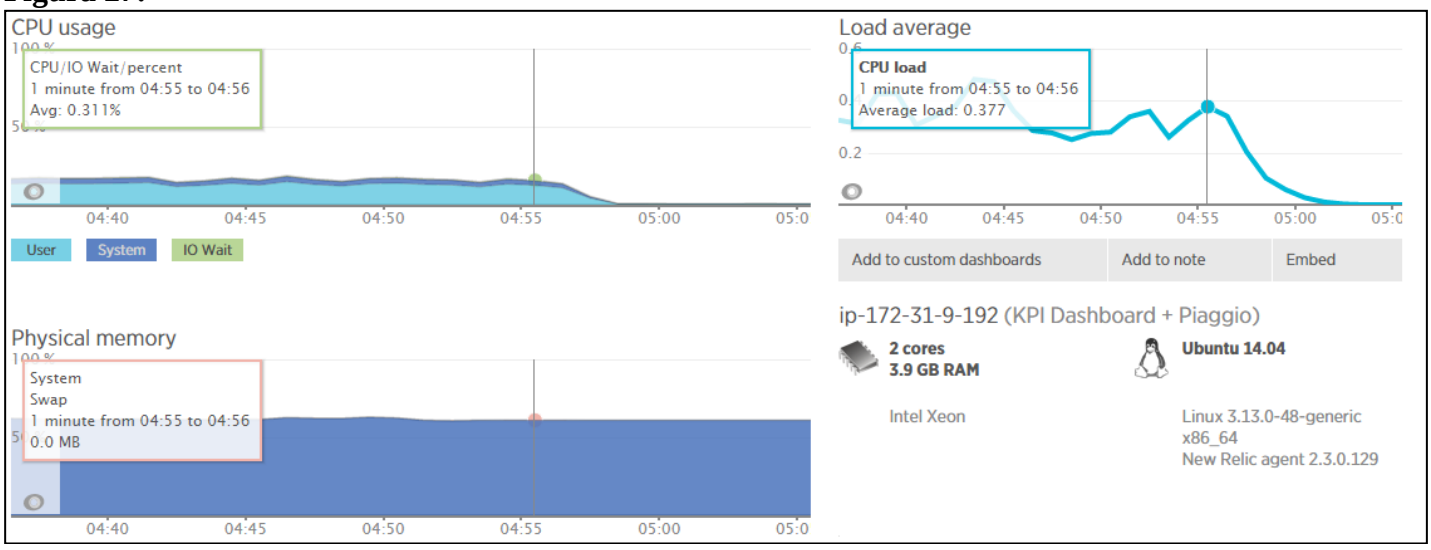
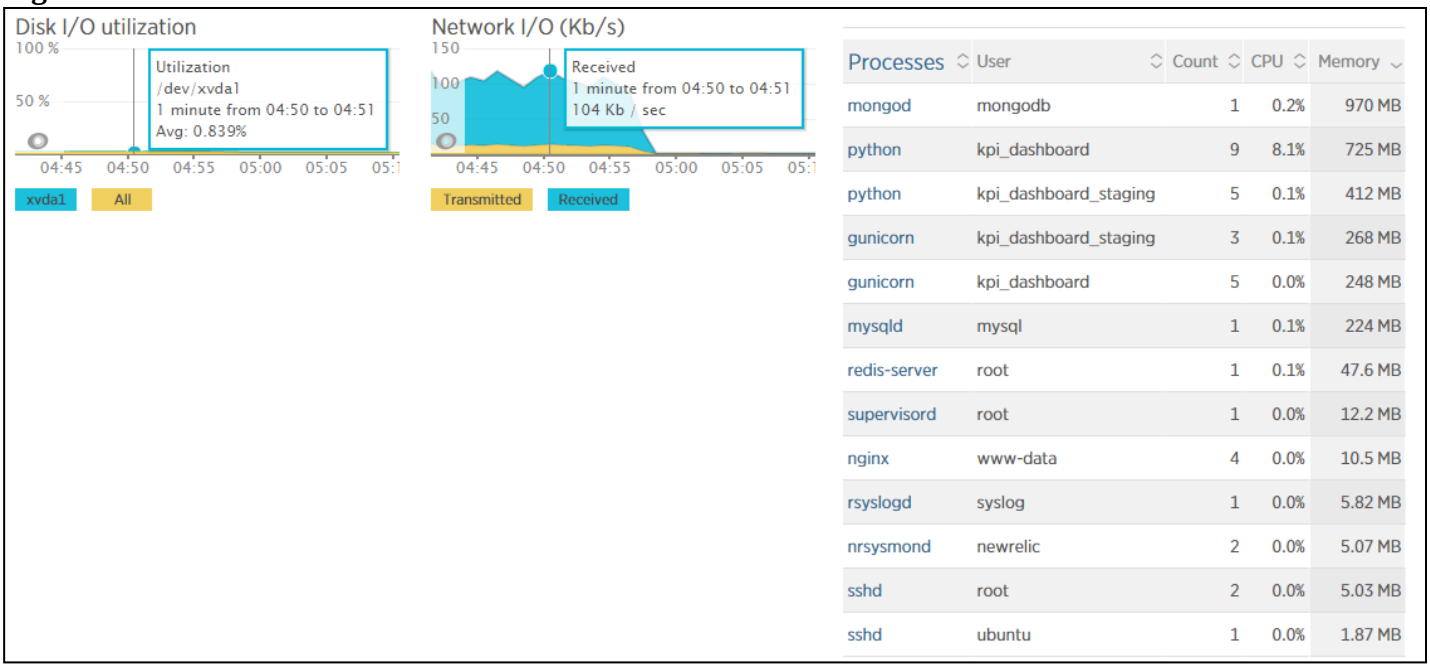


Figura 28:

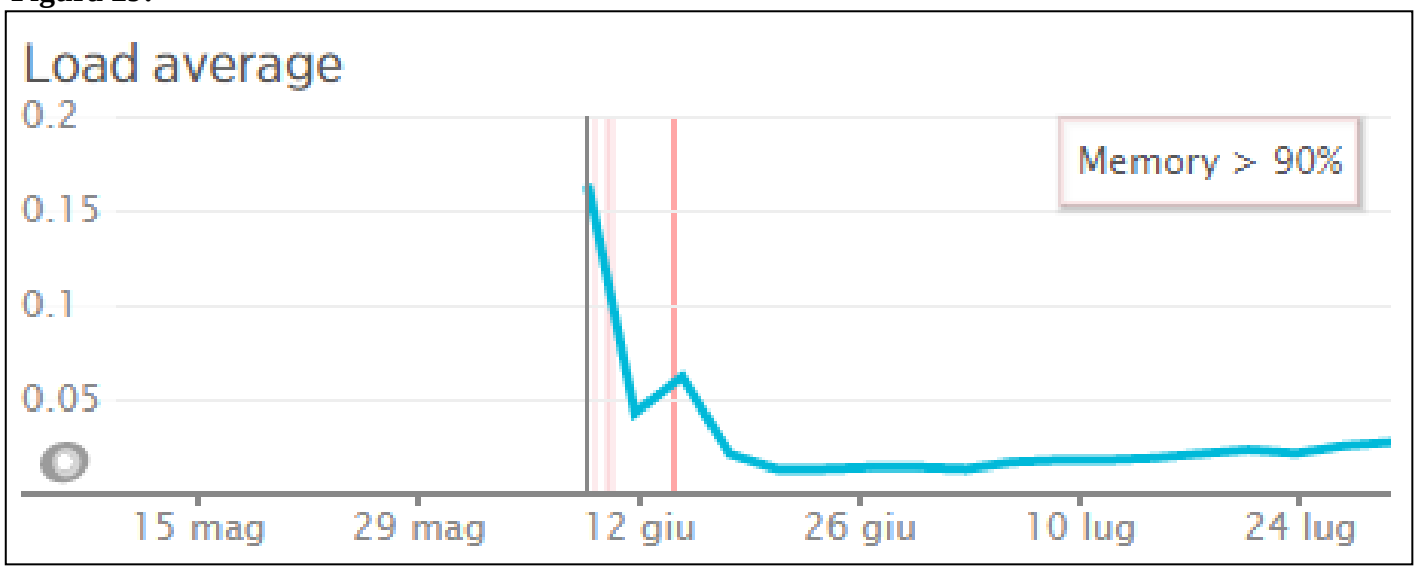


Processes	User	Count	CPU	Memory
mongod	mongodb	1	0.2%	970 MB
python	kpi_dashboard	9	8.1%	725 MB
python	kpi_dashboard_staging	5	0.1%	412 MB
gunicorn	kpi_dashboard_staging	3	0.1%	268 MB
gunicorn	kpi_dashboard	5	0.0%	248 MB
mysqld	mysql	1	0.1%	224 MB
redis-server	root	1	0.1%	47.6 MB
supervisord	root	1	0.0%	12.2 MB
nginx	www-data	4	0.0%	10.5 MB
rsyslogd	syslog	1	0.0%	5.82 MB
nrsysmond	newrelic	2	0.0%	5.07 MB
sshd	root	2	0.0%	5.03 MB
sshd	ubuntu	1	0.0%	1.87 MB

Le rilevazioni hanno permesso di capire la fondamentale importanza dell'individuazione di un giusto bilanciamento tra numero di operazioni eseguite da ciascun worker dell'application server Gunicorn.

In particolare si è richiesto all'applicazione di confrontare i dati su base annuale più volte contemporaneamente (Fig. 29).

Figura 29:



È emerso in maniera evidente come il non imporre un limite al numero di richieste risolte contemporaneamente da ciascun worker, potesse portare a forte stress lato CPU (utilizzo > 90%) con conseguente rischio di downtime.

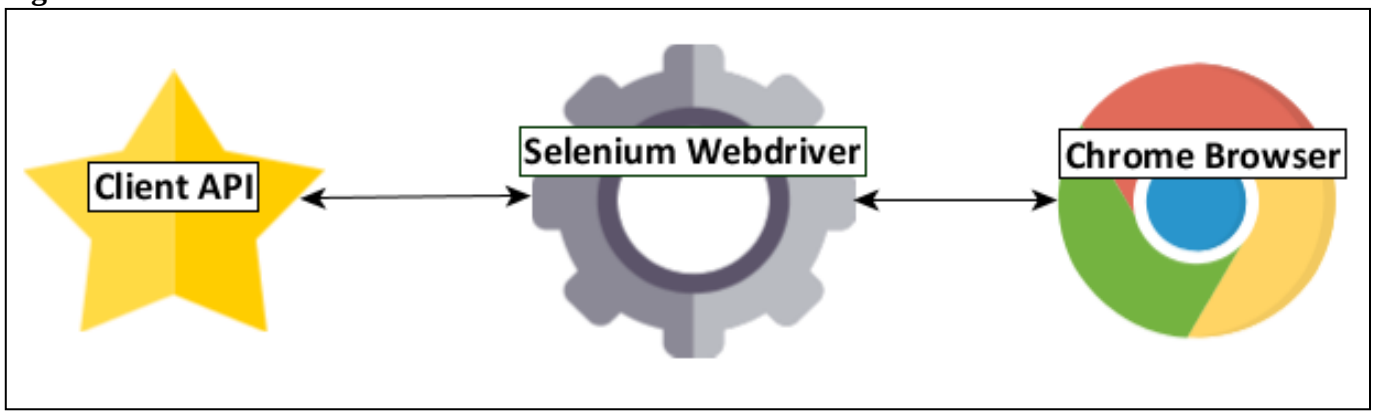
Si è quindi deciso di limitare a 1 il numero di richieste eseguite contemporaneamente e di parallelizzare le richieste optando per un numero di worker pari a 2* numero di core + 1 (ossia 5).

In aggiunta, le statistiche sui processi hanno permesso di capire per esempio che il database non relazionale MongoDB è soggetto ad un aumento del livello di memoria fisica utilizzata, al crescere dei dati memorizzati.
Questa informazione potrebbe portare con sé la necessità futura di spostare MongoDB su servizio esterno, o addirittura all'abbandono di questa tecnologia a favore di uno strumento orientato ai Big Data, come Google Big Query.

5.3 TEST CON SELENIUM

Selenium [51] è un sistema open-source ideato nel 2004 da Jason Huggins per la realizzazione di test automatici Web-based.
Lo strumento, applicabile su qualunque applicazione Web (e conseguentemente browser Web) che supporti JavaScript, permette di creare casi di studio utilizzando semplici script supportati da innumerevoli linguaggi di programmazione, tra cui evidentemente Python. Tali librerie client non sono altro che client http incapsulati nella Selenium client API.
In aggiunta è richiesto l'utilizzo di un WebDriver, componente che accetta comandi da Selenium client API e che proprio a partire da questi comandi inizia direttamente una istanza browser e la controlla (Fig. 30).

Figura 30:



I WebDriver disponibili sono relativi ai browser più popolari (ChromeDriver, FirefoxDriver, InternetExplorerDriver, SafariDriver) e ad altri browser volti alla realizzazione di test più specifici (PhantomJSDriver, RemoteWebDriver).

L'applicazione di Selenium ha permesso di simulare il comportamento umano e della corrispettiva risposta grafica nell'interfaccia della *VAD*, al fine di mostrare bug non evidenti lato backend, ma comunque impattanti in ambito *user interface* (*UI*).
In particolare si è notato come il Webdriver sia estremamente sensibile all'utilizzo di librerie dinamiche AngularJS, dal momento che trattandosi di test case realizzati in modalità programmatica la gestione dell'errore viene lasciata libera.
Per poter dunque utilizzare correttamente Selenium nella navigazione della applicazione è stato creato un semplice metodo per individuare il selettore CSS di un elemento in pagina (bottone, link, grafico..) e, fintanto che la selezione dell'elemento non va a buon fine, il ciclo riparte fino ad un massimo di 100 interazioni.

```
def wait_for_element_clickable(driver, selector):
    number_of_loops = 0
    while True and number_of_loops < 100:
        try:
            driver.find_element_by_css_selector(selector).click()
            time.sleep(3)
            break
        except Exception as e:
            number_of_loops += 1
```

5.4 REAL TIME ERROR TRACKING CON SENTRY

Sentry [52] è un sistema di tracciamento dell'errore in tempo reale per ambienti di produzione. Utilizzato da clienti del calibro di Uber, Airbnb, Dropbox, per citarne alcuni, supporta un vasto numero di linguaggi e configurazioni differenti.
Nel caso della *VAD* è stato installato sul framework Django.
Questo sistema di tracciamento dell'errore è in grado di intercettare tutti gli errori che si presentano su un applicativo lato utente, su ogni dispositivo e ogni piattaforma, fornendo informazioni complete per riprodurre e risolvere agilmente i problemi.
Il test in produzione è infatti un ottimo metodo per esporre l'applicazione a scenari del mondo reale e identificare errori e anomalie non riscontrate in un tradizionale ambiente di test.
L'installazione su framework Django è così ben documentata, che l'utilizzo di soli pochi passaggi permette di attivare Sentry:

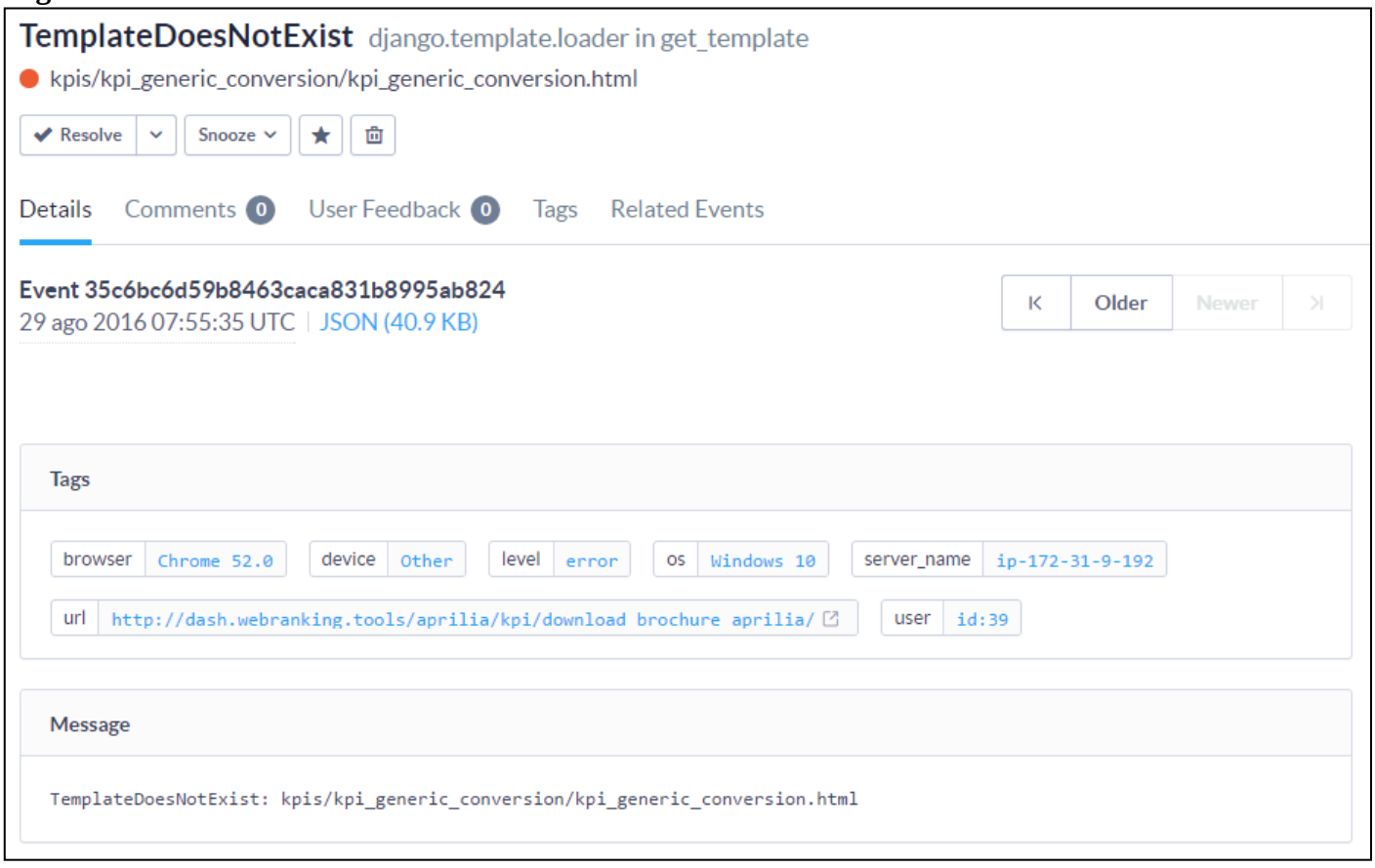
1. creazione del progetto sul portale Sentry;
2. selezione della tipologia di applicazione su cui attivare Sentry;

3. inserimento nelle file di settings di Django della applicazione di supporto a Sentry ("*raven.contrib.django.raven_compat*"), in modo da creare un collegamento tra report e applicazione;
4. aggiunta del dizionario di configurazione del DNS Sentry denominato RAVEN_CONFIG al file di settings di Django.

Caratteristica principale di questo sistema di testing in ambienti di produzione risiede nella possibilità di un riscontro immediato dell'impatto ad ogni nuovo rilascio dell'applicazione. Non secondario appare poi il potere trasparente dato all'utente, nel senso che non si necessita di un feedback diretto degli errori, poiché Sentry tiene traccia di tutti i crash riscontrati durante l'intera *user experience*.

Si riporta un esempio di messaggio di errore (Fig. 31) riscontrato a seguito dell'implementazione del template HTML di un *Kpi*.

Figura 31:



5.5 *TECNICHE DI MIGLIORAMENTO DELLE PRESTAZIONI*

Il miglioramento delle prestazioni della *VAD* è stato focalizzato su più ambiti:

- lato dataset MongoDB, al fine di memorizzare solo i dati strettamente necessari;
- lato richieste scaricamento e aggregazione del dato, al fine di migliorare i tempi di caricamento.

Per quanto riguarda la memorizzazione dei dati su MongoDB sono stati implementati dei filtri di scaricamento per ciascuno dei record *Analytics* della tabella *Kpi*.

In tale modo è stato possibile richiedere dati a Google Analytics filtrati sulle informazioni di interesse di ciascun *Kpi*.

Attraverso un semplice metodo auto costruito che analizza una lista di oggetti JSON è infatti possibile applicare una selezione "fine" a richiesta del dato.

Ad esempio per il *Kpi* relativo all'azione di "Download della Brochure", è stato applicato un filtro del tipo:

```
[
  {
    "values": [
      "download_brochure"
    ],
    "key": "ga:eventCategory"
  },
  {
    "operator": "=~",
    "values": [
      "moto",
      "accessori"
    ],
    "key": "ga:pagePath"
  }
]
```

In sintesi viene specificata una chiave (*key*) relativa ad una *Dimensione*, ed il filtro viene applicato per i valori (*values*) definiti, secondo il parametro di operatore opzionale (*operator*).
Ove non specificato l'operatore assume il valore di uguaglianza stretta, nel secondo oggetto rappresenta invece l'operatore di inclusione (= ~) per cui i risultati sono filtrati se i valori "moto" o "accessori" sono contenuti nella *Dimensione* "ga:pagePath".

Le richieste di scaricamento e aggregazione del dato hanno coinvolto l'utilizzo di tre livelli di caching:

1. lato *Analytics*, è stato scelto di memorizzare su file (in direttorio dedicato) l'intera risposta ottenuta dopo ciascuna richiesta alle API Google, in modo da poter consentire una veloce riscrittura del dato in caso di necessità;
2. lato *SEO*, analogamente è stato scelto di memorizzare su file (in direttorio dedicato) l'intera risposta ottenuta dalle API di Authority Labs;
3. lato richieste REST per i dati relativi a ciascun *Kpi*, è stata scelta la memorizzazione delle risposte tramite il servizio di cache offerto dalla libreria di Django REST Framework. In tal modo una richiesta idem potente di una richiesta precedente, otterrà rapida risposta potendo usufruire della cache già memorizzata. Tale guadagno prestazionale si riflette maggiormente per richieste di dati relativi ad archi temporali di durata mensile ed oltre, con il risultato di una riduzione dei tempi di caricamento di circa l'80%.

5.6 RISULTATI OTTENUTI

Il paragrafo pone accento sul risultato del lavoro svolto sulla *VAD*.
Nell'immagine successiva (Fig. 32) è riportato il form di autenticazione all'applicazione; si ricorda che a ciascun utente sono riservati un numero di brand personalizzati con facoltà di cambiare stile della pagina in base al brand selezionato.
La pagina principale della dashboard (Fig. 33) raccoglie i dati dei *Kpi* attivi per il brand, mostrando inoltre una icona a forma di punto interrogativo sopra ciascuno, che descrive i dettagli sulle caratteristiche del *Kpi* visualizzato.

Figura 32:

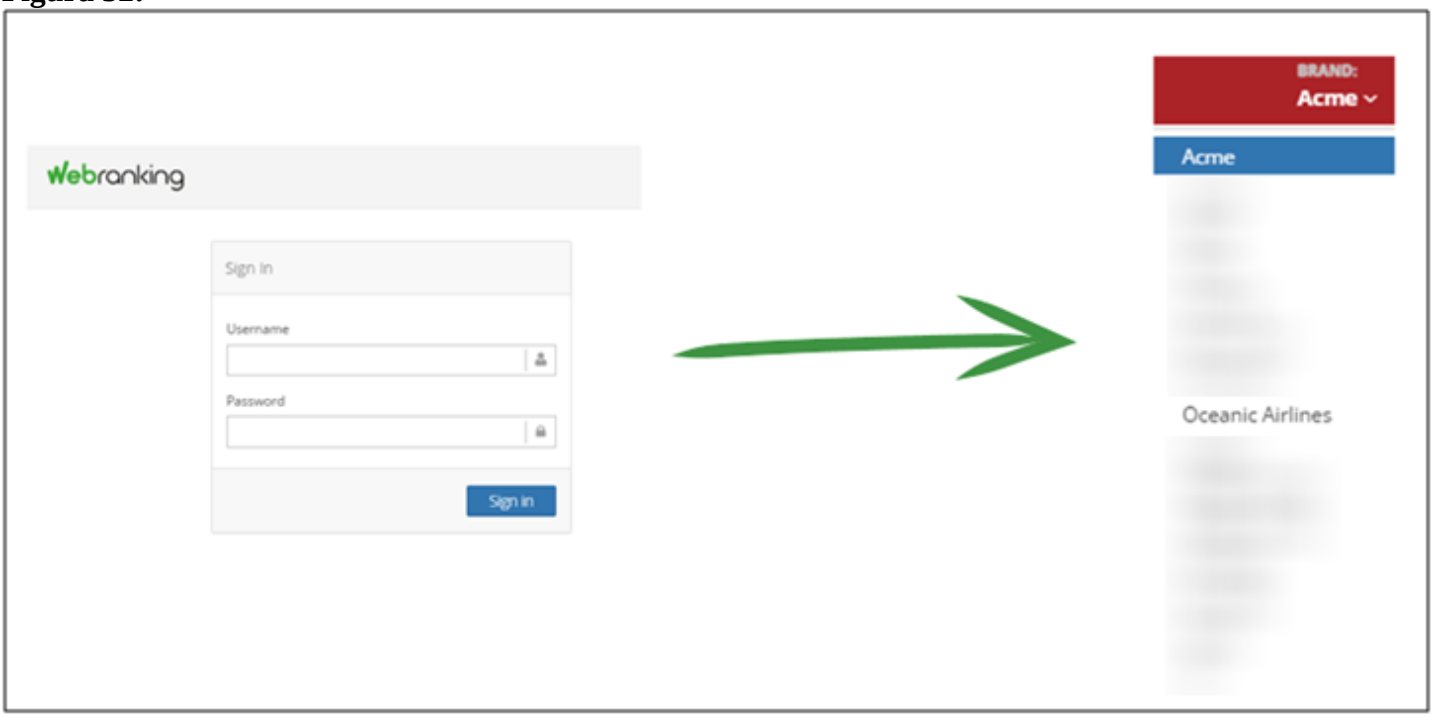
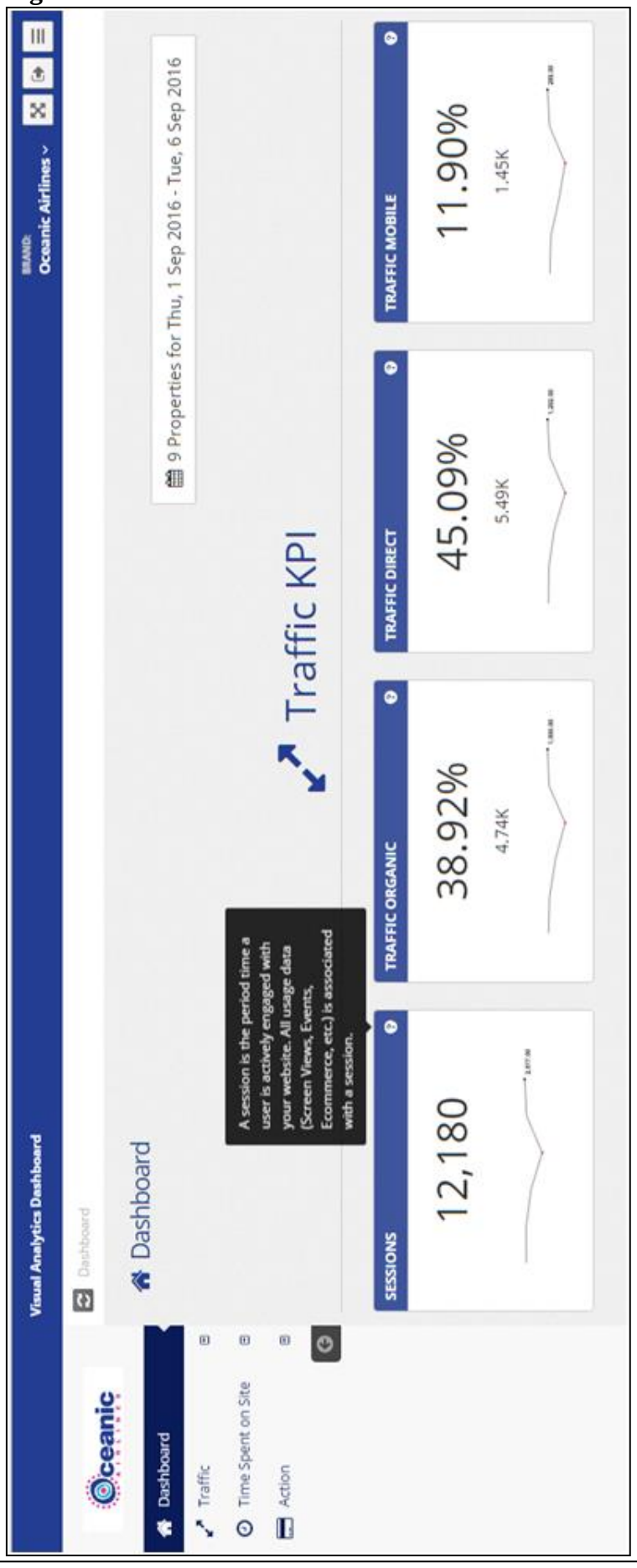

Webranking
Sign In
Username
Password
Sign in
BRAND:
Acme
Acme
Oceanic Airlines

Figura 33:


Visual Analytics Dashboard
BRAND:
Oceanic Airlines
Dashboard
Dashboard
Traffic
Time Spent on Site
Action
Dashboard
9 Properties for Thu, 1 Sep 2016 - Tue, 6 Sep 2016
Traffic KPI
A session is the period time a user is actively engaged with your website. All usage data (Screen Views, Events, Ecommerce, etc.) is associated with a session.
SESSIONS
12,180
TRAFFIC ORGANIC
38.92%
4.74K
TRAFFIC DIRECT
45.09%
5.49K
TRAFFIC MOBILE
11.90%
1.45K

Ora è mostrato il dettaglio di pagina del *Kpi* che comprende i dati delle sessioni totali degli utenti (Fig. 34), con particolare riferimento ai differenti tipi di grafico riportati (Fig. 35).

Figura 34:

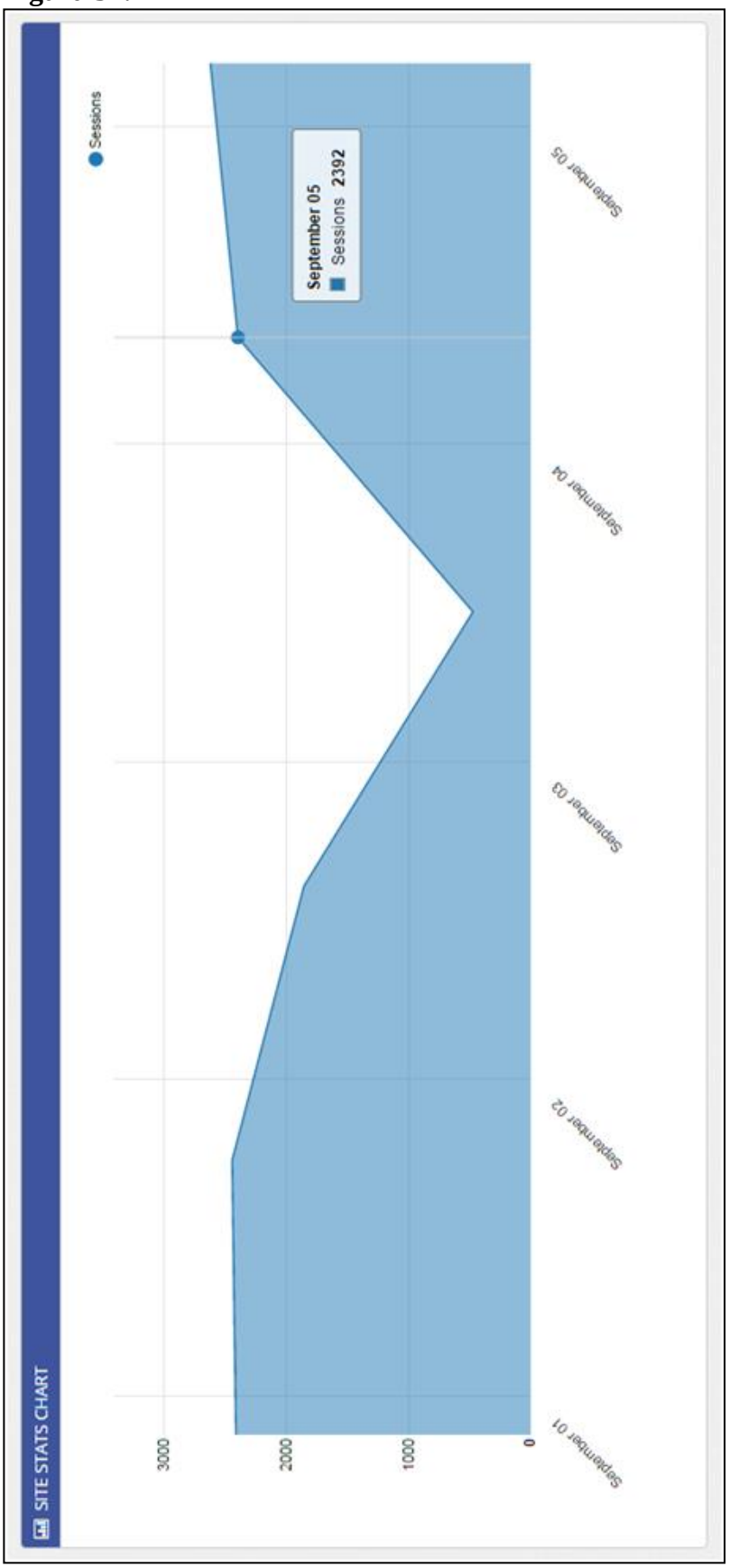
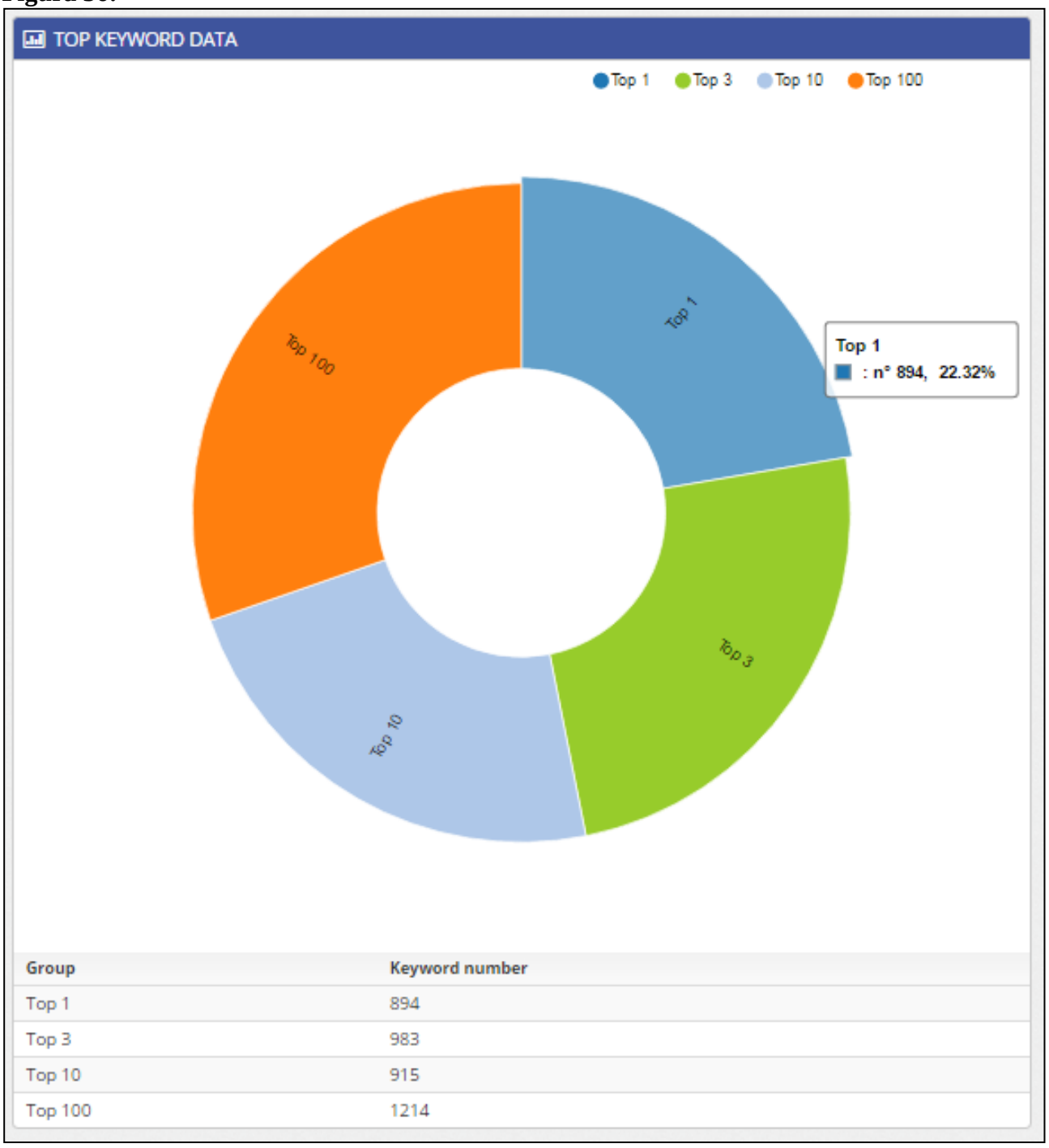


Figura 35:

MAP
Italy: 19584
0
92.71K
Property
Sessions
%
Russian Federation
92.71K
51.01%
Italy
19.58K
10.78%
Turkey
11.72K
6.45%
Middle East (AR)
8.94K
4.92%
United Kingdom
8.86K
4.88%
5 - 28
Load More
Show All
DEVICES
Mobile
Tablet
Desktop
Mobile
Tablet
Desktop
Device
Sessions
%
Desktop
85.19K
46.87%
Mobile
82.61K
45.45%
Tablet
13.94K
7.67%

Nello specifico, riporto la pagina (Fig. 36) dei risultati riguardati il posizionamento delle *keyword* ripartite in base ai dati del numero di *keyword* in top posizionamento (top 1, top3 top 10 e top100), e a seguire il relativo grafico a granularità mensile (Fig. 37) nonché l'elenco risultati in base a singola parola chiave e categorie (Fig. 38).

Figura 36:



Group	Keyword number
Top 1	894
Top 3	983
Top 10	915
Top 100	1214

Figura 37:

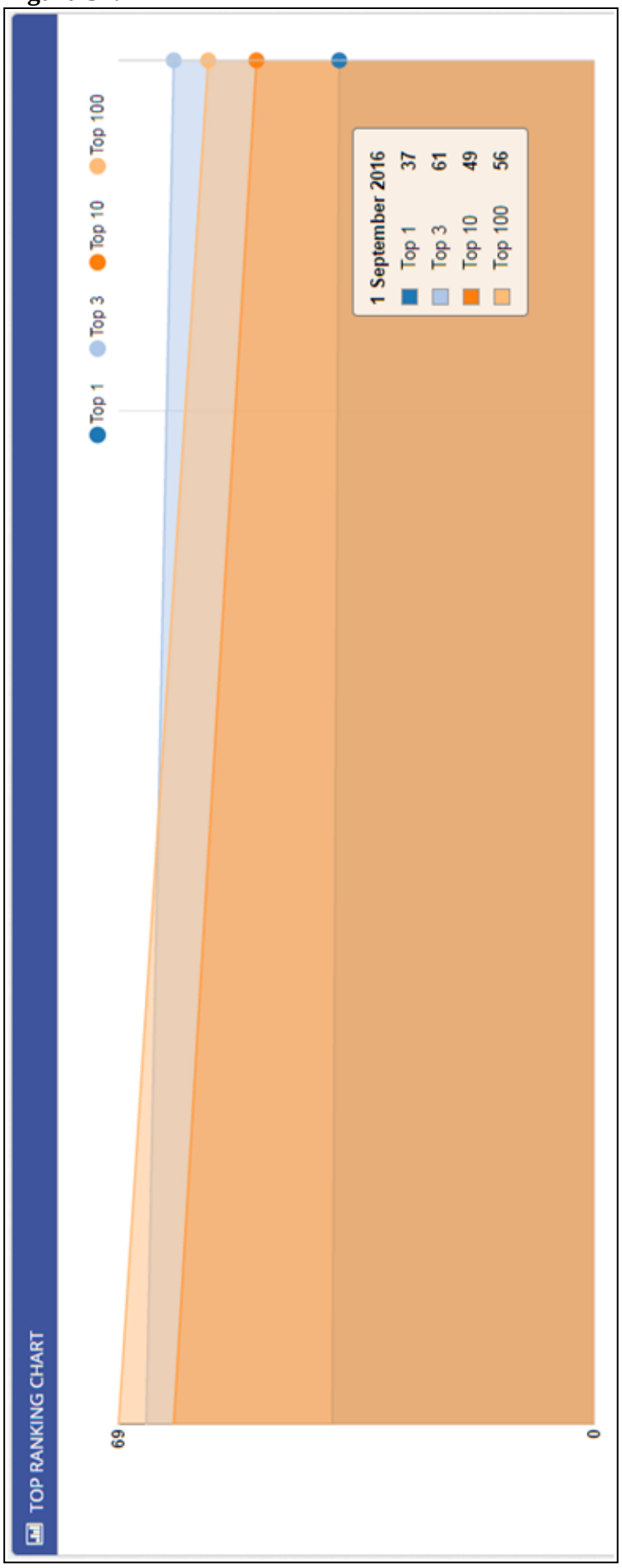

TOP RANKING CHART
Top 1
Top 3
Top 10
Top 100
1 September 2016
Top 1 37
Top 3 61
Top 10 49
Top 100 56
69
0

Figura 38:

RANKING KEYWORD
CATEGORY
Keyword
URL
Ranking
Volume
Category
Top 1
Top 3
Top 10
Top 100
Load More
Show All
12 - 39
4 - 4

In ultimo è mostrato il dettaglio della funzionalità di comparazione rispettivamente per periodo (Fig. 39) e per viste distinte (Fig. 40).

Figura 39:

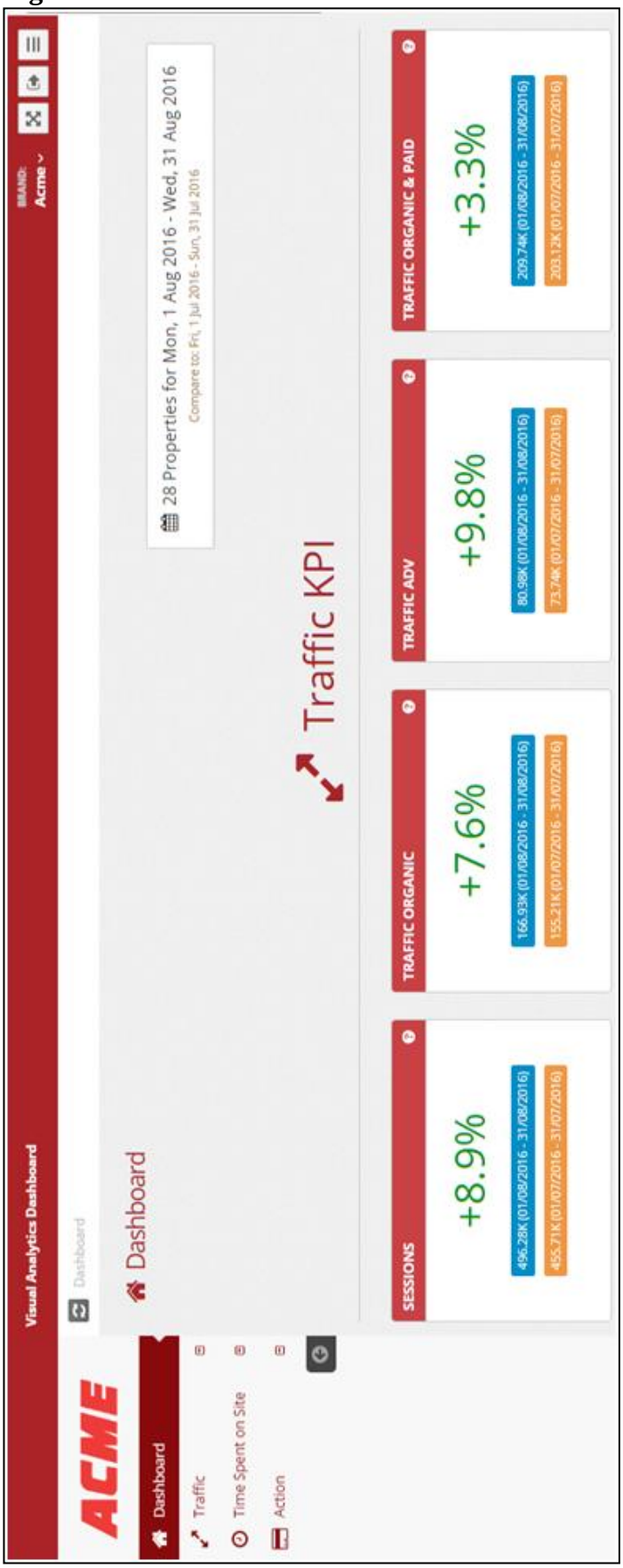
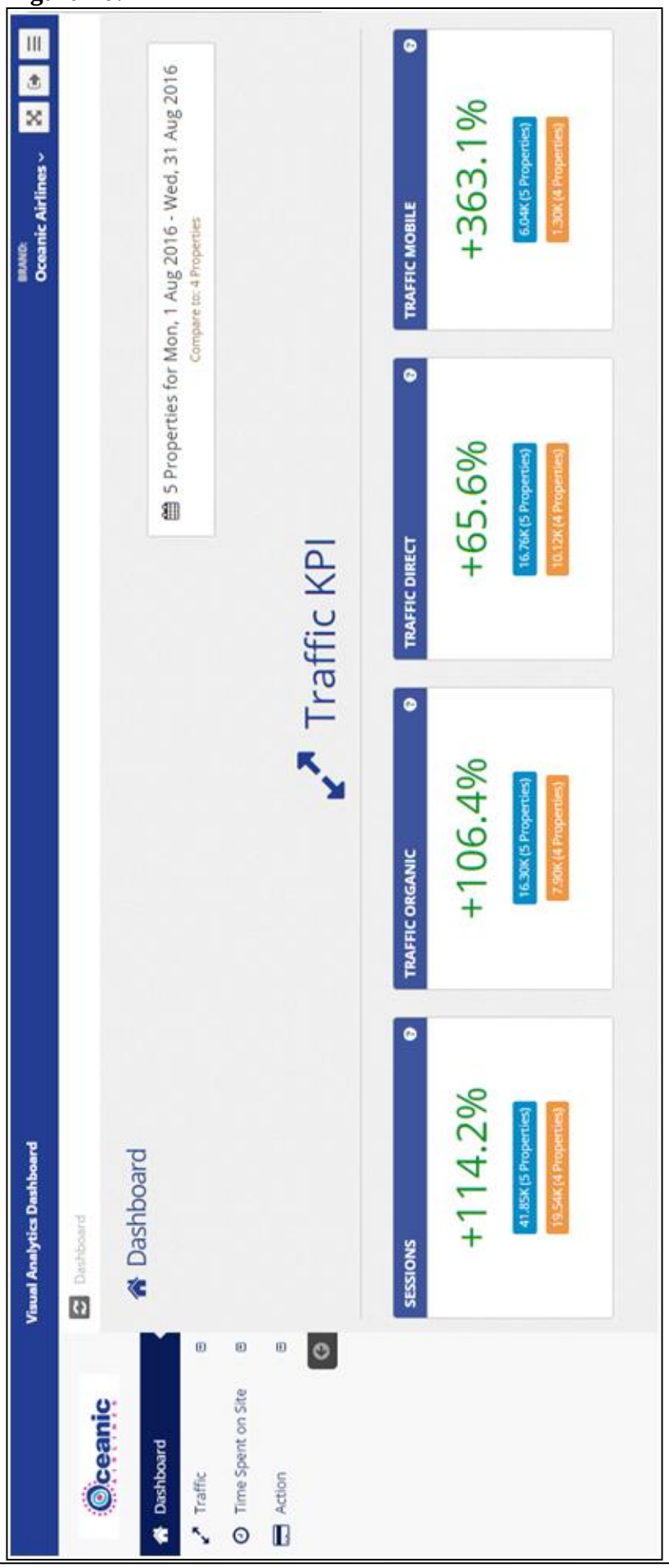
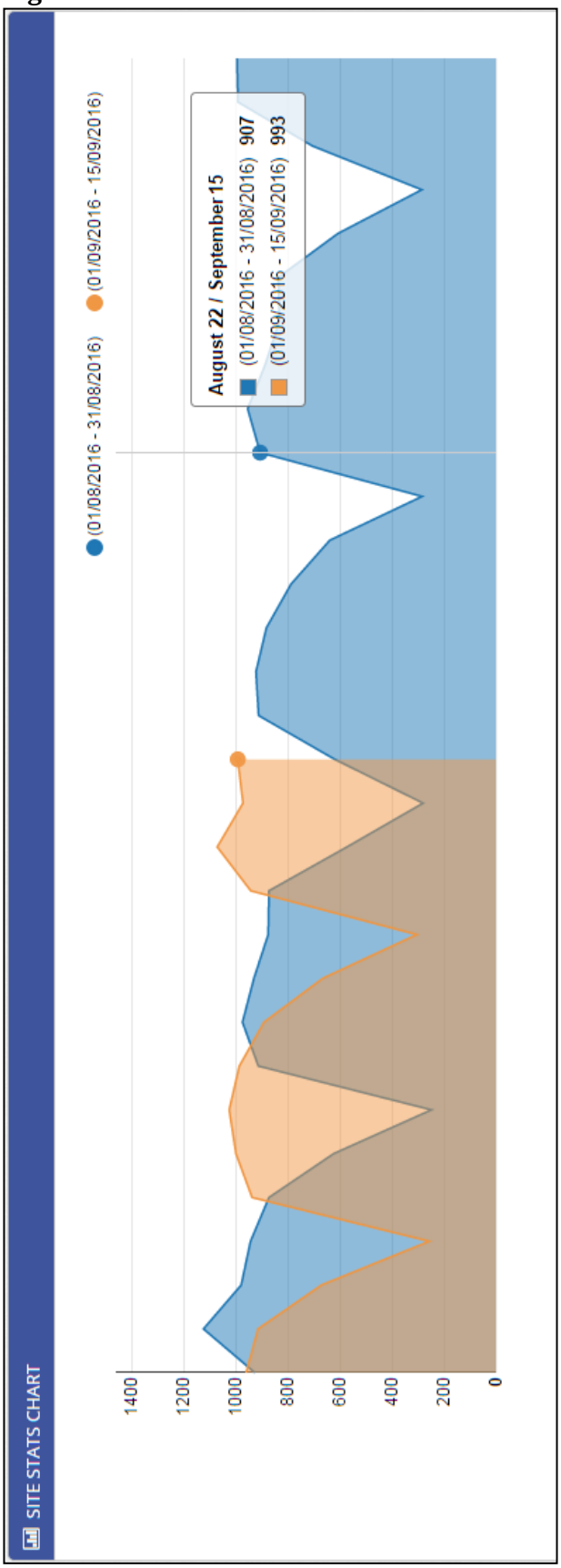


Figura 40:


Visual Analytics Dashboard
BRAND:
Oceanic Airlines
Dashboard
Dashboard
Traffic
Time Spent on Site
Action
Dashboard
5 Properties for Mon, 1 Aug 2016 - Wed, 31 Aug 2016
Compare to: 4 Properties
Traffic KPI
SESSIONS
+114.2%
41.85K (5 Properties)
19.54K (4 Properties)
TRAFFIC ORGANIC
+106.4%
16.30K (5 Properties)
7.90K (4 Properties)
TRAFFIC DIRECT
+65.6%
16.76K (5 Properties)
10.12K (4 Properties)
TRAFFIC MOBILE
+363.1%
6.04K (5 Properties)
1.30K (4 Properties)

Si noti il particolare della comparazione tra periodi temporali di lunghezza non omogenea nel grafico a granularità giornaliera della pagina di dettaglio del *Kpi* riguardate gli utenti provenienti da traffico organico (Fig. 41).

Figura 41:



6 CONCLUSIONI E SVILUPPI FUTURI

Nell'ambito del tirocinio presso la Netidea Webranking s.r.l. di Correggio è stato studiato a fondo il processo di analisi e ottimizzazione di un sito Web, affrontando un caso di sviluppo concreto inerente i siti dei clienti committenti. L'implementazione dell'applicazione ha permesso di comprendere che la fase di ottimizzazione del processo risulta essere di fondamentale importanza per il settore del Webmarketing.

La tesi di laurea ha affrontato un campo relativamente nuovo della Search Marketing, quali la creazione di dashboard ad hoc per la visualizzazione dei dati ottenuti da tracciamenti *Web Analytics* e da posizionamenti di parole chiave nei motori di ricerca, al fine di individuare le principali problematiche e le possibili soluzioni da adottare.

Tra le sfide affrontate, non si dimentichi la parte che ho dedicato allo studio formativo e tecnologico necessario alla progettazione e realizzazione dell'applicazione.

Nel dettaglio, il lavoro iniziale ha contemplato la fase di formazione relativa al background teorico delle discipline di *Web Analytics* e *SEO*. Durante la ricerca, particolare attenzione è stata posta alla scelta delle caratteristiche dei vari tipi di dato da prendere in esame, nonché ai requisiti specifici e alle limitazioni intrinseche degli strumenti disponibili attualmente sul mercato.

Sono stati individuati due servizi, Google Analytics ed Authority Labs, per consentire lo scaricamento e la fruizione dei dati di interesse.

Successivamente, l'attenzione volta alle peculiari esigenze dei clienti ha portato alla scoperta di problematiche in vari ambiti con conseguente necessità di valutazione dei possibili approcci atti a ottenere la soluzione ottimale e la soddisfazione del committente.

Per esempio la strategia implementata per evitare il campionamento dei dati ha consentito aggregazione e confronti puri su archi temporali senza limiti di durata, cosa che un approccio di scaricamento in tempo reale, quale da piattaforma di Google Analytics, non contempla.

Dall'analisi delle statistiche dei leader attualmente presenti sul mercato si è scelto Amazon Web Services come infrastruttura di servizio di hosting per l'applicazione.

Si è così giunti alla fase di progettazione e realizzazione dell'architettura dell'applicazione, focalizzando l'attenzione principalmente allo sviluppo di 3 livelli logici fondamentali e delle possibili soluzioni tecnologiche per l'implementazione del progetto.

In ultimo dopo aver dedicato ampio spazio all'organizzazione del lavoro e alla fase di testing su più livelli, è stata rilasciata la Visual Analytics Dashboard (*VAD*), e da qui il processo di integrazione continua inerente lo studio di tecniche di miglioramento delle prestazioni ed il tracciamento in tempo reale dei bachi in ambiente di produzione.

Il contributo al lavoro svolto ha permesso di produrre una applicazione con le seguenti caratteristiche:

- altamente performante, in grado quindi di processare e analizzare ingenti insiemi di dati in un quantitativo di tempo relativamente breve;
- versatile, in quanto capace di elaborare dati eterogenei provenienti da fonti diverse;
- modulare, in quanto l'approccio di sviluppo applicato permette una rapida integrazione di componenti per lo scaricamento del dato da nuove piattaforme;
- efficiente nella presentazione di report multi-brand e multi-sito con alto grado di personalizzazione grafica e filtro sul dato;
- autosufficiente, intesa come proprietà di scaricare e rinfrescare il dato dei clienti in completa autonomia.

6.1 SVILUPPI FUTURI

Gli sviluppi futuri riguardano vari processi volti a ottimizzazione, testing e aggiunta di nuove funzionalità.

L'ottimizzazione potrebbe partire dallo spostamento dei dati memorizzati su MongoDB, che ricordo essere estremamente avido in termini di risorse e spazio occupato, su un sistema orientato invece ai Big Data, tipo Google Big Query [53], un *pay per use* di *Query as a Service* (*QaaS*) che prevede l'esecuzione di interrogazioni tramite linguaggio SQL-like nel cloud.

Big Query si basa su vari livelli di astrazione retti da un sistema di memorizzazione estremamente scalabile [54].

Da un lato quindi scalabilità a livello di memoria, e dall'altro scalabilità in termini di linguaggio di interrogazione, potendo contare su sintassi simil-SQL.

Quest'ultima tendenza è proprio la direzione che l'intera industria sta prendendo attualmente, il che appare piuttosto ironico considerato che query di sistemi NoSQL facciano uso di sintassi SQL.

Tuttavia la pervasività del linguaggio SQL ha di fatto creato uno standard, che difficilmente potrà essere superato nel breve periodo.

Ulteriore possibilità risiederebbe nel parallelizzare la computazione tramite l'utilizzo di Spark [55], framework open-source, sviluppato presso l'AMPlab dell'Università di California e successivamente donato alla Apache Software Foundation.

Questo framework esegue gran parte dell'elaborazione all'interno della memoria RAM attraverso le cosiddette Resilient Distributed Datasets (RDD), interfacce trasparenti in termini di occupazione di memoria e risorse, recuperabili in caso di errore [56].

Sempre lato ottimizzazione, annovero la soluzione scaturita e quindi adottata al fine di aggirare i bug che influenzano i dati di Google Analytics non ancora consolidati, che ha previsto una procedura di rinfrescamento dati con cadenza mensile.
Questa funzionalità non è comunque esente da possibilità di errore, come nel caso in cui un baco sia segnalato dopo vari mesi di attesa.
In quest'ottica risulta evidente la difficoltà di dover provvedere al rinfrescamento di tutti i dati dei *Kpi Web Analytics* per archi temporali elevati dato che, come evidenziato, lo scaricamento necessita di richieste API pari al numero di siti moltiplicato per i relativi *Kpi*.
Nondimeno, la procedura di rinfrescamento su lunghi archi temporali potrebbe facilmente generare tempi altrettanto lunghi di disservizio.
Per ovviare a questo inconveniente, la soluzione proposta prevede uno storage principale (*master storage*) e uno secondario (*slave storage*) precedentemente copiato dal principale.
Le operazioni di rinfrescamento dei dati sono eseguite su *slave* storage, e garantiscono quindi continuità di servizio applicativo (su *master storage*), in quanto è solo al termine della procedura di aggiornamento che viene attuato lo scambio tra *slave* e *master storage*.

La fase di testing, con particolare riferimento alla pre-produzione su Windows antecedente al rilascio, potrebbe essere significativamente migliorata adottando Docker [57], progetto open-source che automatizza il rilascio di applicazioni garantendo la possibilità di replica di un ambiente di test/produzione.
Essenzialmente Docker impacchetta un sistema Linux, scelto in base alla versione del server in produzione, in piccole macchine virtuali (*container)* ognuna della quali contiene un insieme di programmi e servizi ben isolati dal resto. I *container*, contenenti da un pool di applicazioni fino ad un singolo servizio, sono realizzati per essere portabili, potendo essere "spediti" da un sistema all'altro come veri e propri container navali.

Per quanto concerne l'aggiunta di nuove funzionalità nella *VAD*, si può esaminare l'inserimento di tecniche di machine learning di tipo previsionale.
Valutando infatti l'andamento storico dei dati *Web Analytics* si potrebbero implementare algoritmi volti alla predizione delle sessioni utente future, tenendo conto anche dei possibili investimenti in termini di annunci pubblicitari.
Analogamente lato *SEO*, potrebbero essere predisposte previsioni riguardo l'andamento delle *keyword* tenendo conto per esempio dei nuovi competitor presenti nei motori di ricerca.
Queste tecniche potrebbero essere implementate utilizzando Scikit-learn [58], libreria Python open-source, contenente vari algoritmi per l'applicazione di modelli di machine learning.

APPENDICE

ELUSIONE DEL CAMPIONAMENTO

La procedura autodefinita *unsampling_data_as_df*, è attivata ogni qual volta la risposta ricevuta da Google Analytics restituisce la chiave "containsSampledData" con valore "True".

Il metodo accetta in input un dizionario di parametri per effettuare due nuove interrogazioni API basate sulla divisione del periodo di analisi, agendo ricorsivamente nei casi in cui si ripresenti il fenomeno di campionamento.

Riporto ora la definizione:

```
def unsampling_data_as_df(self, query_params):

        logger.info('Unsampling GA con parametri:
{}'.format(query_params))

        date_delta = (query_params['end_date'] -
query_params['start_date']).days

        query_params_start = self.query_params(
            id_view=query_params['id_view'],
            start_date=query_params['start_date'],
            end_date=query_params['start_date'] +
timedelta(days=date_delta/2),
            metrics=query_params['metrics'],
            dimensions=query_params['dimensions'],
            filter_mask=query_params['filter_mask']
        )
        query_params_end = self.query_params(
            id_view=query_params['id_view'],
            start_date=query_params['start_date'] +
timedelta(days=date_delta/2 + 1),
            end_date=query_params['end_date'],
            metrics=query_params['metrics'],
            dimensions=query_params['dimensions'],
            filter_mask=query_params['filter_mask']
        )

        result_start = {'nextLink': True}
        start_index = 1
        max_results = 10000  # max è 10000

        while 'nextLink' in result_start.keys():

            params = query_params_start

            params.update({
```

```
                'start_index': str(start_index),
                'max_results': str(max_results),
            })

            logger.info("SPLIT Request GA: {}".format(params))

            result_start =
self.client.data().ga().get(**params).execute()

            if result_start.get('totalResults', 0) == 0:
                result_start.update({'rows': []})
                result_start.update({'containsSampledData':
False})

            start_index += max_results

        if result_start['containsSampledData'] and date_delta !=
1:
            temporary_params = dict(query_params)
            temporary_params['end_date'] =
query_params['start_date'] + timedelta(days=date_delta/2)

            result_start['rows'] =
self.unsampling_data_as_df(temporary_params)

        result_end = {'nextLink': True}
        start_index = 1
        max_results = 10000

        while 'nextLink' in result_end.keys():

            params = query_params_end

            params.update({
                'start_index': str(start_index),
                'max_results': str(max_results),
            })

            logger.info("SPLIT Request GA: {}".format(params))

            result_end =
self.client.data().ga().get(**params).execute()

            if result_end.get('totalResults', 0) == 0:
                result_end.update({'rows': []})
                result_end.update({'containsSampledData': False})

            start_index += max_results

        if result_end['containsSampledData'] and date_delta != 1:
            temporary_params_1 = dict(query_params)
            temporary_params_1['start_date'] =
query_params['start_date'] + timedelta(days=date_delta/2 + 1)
```

```
            result_end['rows'] =
self.unsampling_data_as_df(temporary_params_1)

        for element in result_start['rows']:
            result_end['rows'].append(element)

        return result_end['rows']
```

RINFRESCAMENTO DEL DATO

La procedura autodefinita *refresh_pd_to_mongo*, accetta come parametri un pandas.DataFrame di dati rinfrescati tramite chiamata API ed effettua una pulizia nella collezione MongoDB, record dopo record:

```
def refresh_pd_to_mongo(coll, df):
        """ Refresh Mongo collection
        """

        records = df.to_dict(orient="records")

        for record in records:
            coll.find_one_and_delete(
                  {'id_view': record['id_view'], 'ga:date':
record['ga:date']}
            )
            coll.find_one_and_update(
                filter=record,
                update={
                    '$setOnInsert': record
                },
                upsert=True,
                new=True
            )
```

ANALISI DEI MODELLI

Models è la cartella che contiene i modelli relativi alle classi di tabelle del database MySQL:

- *Utente* per la memorizzazione dei dati degli utenti
  1. data di creazione;
  2. nome utente;
  3. password;
  4. email;

5. is_staff, come booleano che stabilisce i diritti dell'utente alla pagina di *Admin*;
6. clienti, come chiave esterna di riferimento del tipo "molti a molti" alla tabella *UtenteCliente* dei clienti associati all'utente;
7. top_keyword_user, come booleano che stabilisce se l'utente ha accesso alle sole keyword segnalate come top.

- *UtenteCliente per la memorizzazione delle associazioni tra ciascun utente e i suoi clienti:*

1. utente, come chiave esterna all'utente associato al cliente;
2. cliente, come chiave esterna al cliente associato all'utente.

- *Cliente* per la memorizzazione dei dati relativi ai clienti:

1. nome;
2. string_id, come riferimento univoco al cliente, utilizzato per mappare la navigazione dei dati del cliente nella dashboard tramite URL;
3. gruppo_cliente, se specificato raccoglie il riferimento testuale al macrogruppo di più clienti;
4. kpis, come chiave esterna di riferimento del tipo "molti a molti" alla tabella *Kpi* attivati per il cliente;
5. css, come riferimento al file css personalizzato per il cliente;
6. logo, come riferimento all'immagine del logo del cliente;
7. mongo_store,come riferimento al database NoSQL MongoDB che contiene i dati di ciascun *Kpi*;
8. source_platform_accounts, come chiave esterna di riferimento alla tabella *SourcePlatformAccount* contentente le chiavi di accesso alle API per lo scaricamento dei dati Analytics e Seo.

- *View* per la memorizzazione dei dati relativi alle viste dei clienti:

1. nome;
2. clienti, come chiave esterna di riferimento di tipo molti a molti alla tabella dei *Clienti*, in quanto ciascuna vista potrebbe essere associata a più di un cliente;
3. id_view, come identificativo della vista di Google Analytics;
4. kpis, come chiave esterna di riferimento di tipo molti a molti alla tabella *ViewKpi* attivati per il cliente;

5. gruppo, come chiave esterna di riferimento di tipo molti a molti alla tabella *ViewGruppo_View* che associata a ciascuna vista una serie di gruppi.

- *ViewKpi* per la memorizzazione dei dati relativi alle associazioni tra *View* e *Kpi*:
  1. kpi, come chiave esterna riferita alla tabella *Kpi*;
  2. view, chiave esterna riferita alla tabella *View*.

- *ViewKpiMetrica* per la memorizzazione dei dati relativi ai *Kpi* con metriche mutabili nel tempo:
  1. nome;
  2. metrica, come chiave esterna riferita alla tabella *Metrica*;
  3. view_kpi, come chiave esterna riferita alla tabella *ViewKpi*.

- *ViewKpiMetricaIntervallo* per la memorizzazione dei dati relativi alle variazioni di ciascun *ViewKpi* nel tempo:
  1. nome;
  2. dal, come data di riferimento iniziale;
  3. al, come data di riferimento finale;
  4. view_kpi, come chiave esterna riferita alla tabella *ViewKpi*.

- *SourcePlatform* per la memorizzazione dei dati relativi ai piattaforme di scaricamento dei dati:
  1. nome;
  2. class_name, come riferimento alla classe Python utilizzata per le richieste dei dati tramite API.

- *SourcePlatformAccount* per la memorizzazione dei dati relativi agli accessi API riferiti ai record della tabella *SourcePlatform*:
  1. nome;
  2. source_platform, chiave esterna riferita alla tabella *SourcePlatform*;
  3. developer_token, come token di accesso;
  4. app_id, come id univoco pubblico della applicazione associata all'account API della *SourcePlatform*;

5. app_secret, come id univoco segreto della applicazione associata all'account API della *SourcePlatform*;
6. logo, come riferimento all'immagine del logo del cliente;
7. refresh_token, come token di aggiornamento necessario per la corretta interrogazione API della *SourcePlatform*.

- *CategoriaKpi* per la memorizzazione dei dati relativi alle categorie di ciascun *Kpi*:

1. nome;
2. fontawesome_icon, come riferimento all'icona FontAwsome caricata dinamicamente nell'interfaccia della applicazione;
3. order, come valore intero che se specificato definisce l'ordine di caricamento delle categorie e dei relativi *Kpi* in pagina.

- *Metrica* per la memorizzazione dei dati relativi alle metriche:

1. nome, come identificativo univoco della metrica utilizzata per lo scaricamento dei *Kpi* Analytics tramite API;
2. proxy, se specificato attribuisce una maschera alla metrica e permette l'associazione tra *Kpi* e *ViewKpiMetrica* definendo la variabilità della metrica nel tempo.

- *Dimensione* per la memorizzazione dei dati relativi alle dimensioni:

1. nome, come identificativo univoco della dimensione utilizzata per lo scaricamento dei *Kpi Analytics* tramite API.

- *Kpi* per la memorizzazione dei dati relativi ai *Kpi*:

1. nome;
2. class_name, come riferimento alla classe Python utilizzata per le tecniche di aggregazione e presentazione dei dati relativi al *Kpi*;
3. categoria, come chiave esterna riferita alla tabella *CategoriaKpi*;
4. chiave, come riferimento univoco al *Kpi*, utilizzato per mappare la navigazione dei dati del *Kpi* nella dashboard tramite URL;
5. [percentauale, secondi, intero, float], come riferimento al tipo di dato relativo al *Kpi*;
6. fontawesome_icon, come riferimento all'icona FontAwsome caricata dinamicamente nell'interfaccia della applicazione;

7. dashboard, come booleano che stabilisce la visibilità del *Kpi* nella homepage dell'applicazione;
8. seo, come booleano che stabilisce se il *Kpi* è di tipo *SEO*;
9. order, come valore intero che se specificato definisce l'ordine di caricamento delle categorie e dei relativi *Kpi* in pagina;
10. metriche, come chiave esterna di riferimento del tipo "molti a molti" alla tabella *Metrica* che raccoglie le metriche identificative del *Kpi*;
11. dimensioni, come chiave esterna di riferimento del tipo "molti a molti" alla tabella *Dimensione* che raccoglie le dimensioni identificative del *Kpi*;
12. extra_data, come formato JSON che semplifica l'aggregazione dei dati del *Kpi*;
13. filter_mask, come formato JSON che filtra scaricamento dei dati del *Kpi* tramite API in base alle caratteristiche del *Kpi* (metriche e dimensioni);
14. source_platform, chiave esterna riferita alla tabella *SourcePlatform*;
15. mongodb_store, come riferimento alla collezione dello store MongoDb che raccoglie i dati scaricati del *Kpi*;
16. help_text, come testo caricato come tooltip di aiuto presentato nell'applicazione su ogni *Kpi*.

- *TipiGruppo* per la memorizzazione dei gruppi riferiti alle vista della tabella *View*:

  1. nome, come identificativo del tipo di gruppo (esempio 'Brand' oppure 'Country').

- *ViewGruppo* per la memorizzazione dei gruppi riferiti alle vista della tabella *View*:

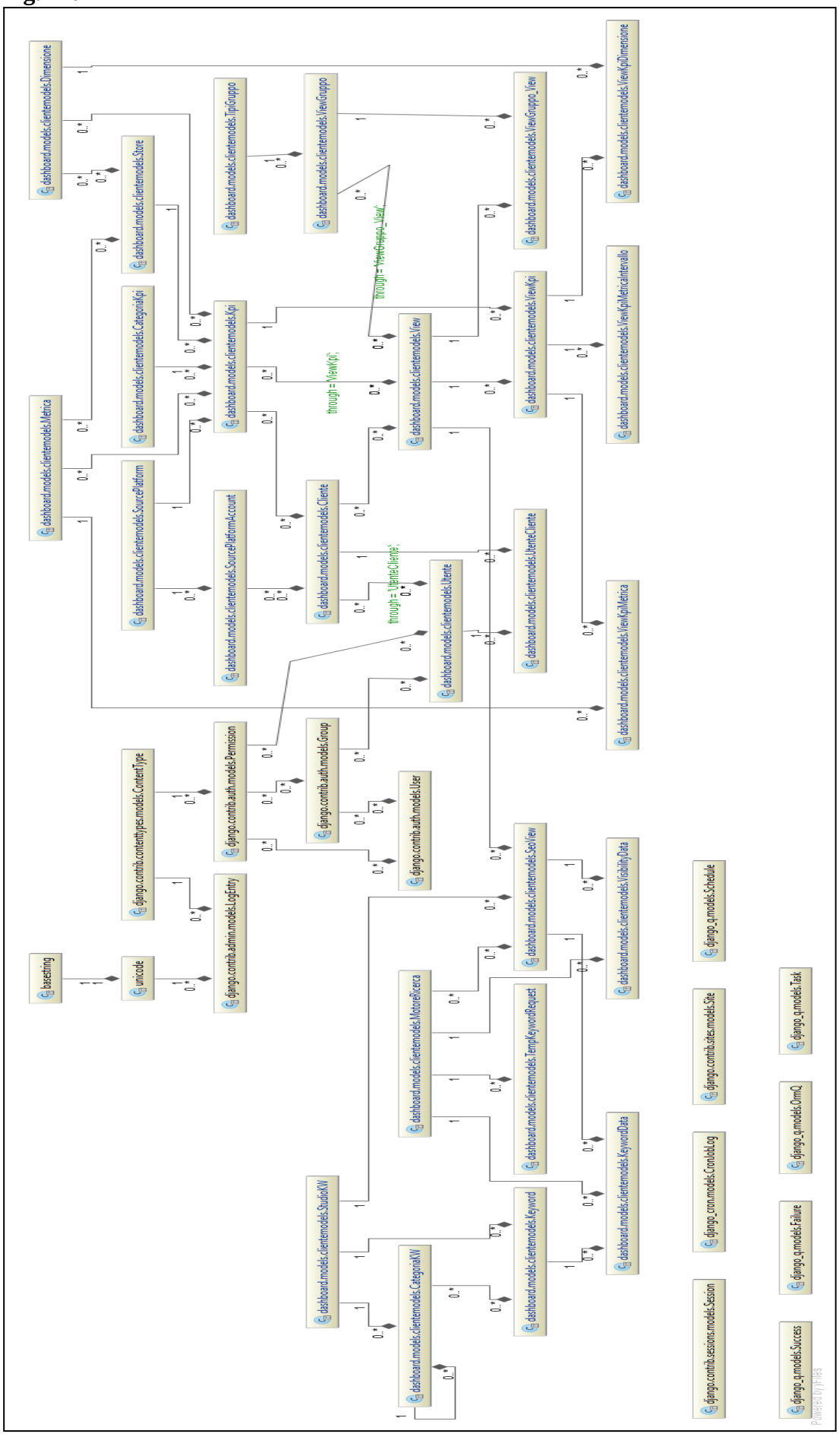
  1. nome, come nome identificativo del gruppo;
  2. tipo, chiave esterna riferita alla tabella *TipiGruppo*.

- *ViewGruppo_View* per la memorizzazione delle associazioni tra gruppi della tabella *ViewGrupo*e e le viste della tabella *View*:

  1. gruppo, chiave esterna riferita alla tabella *ViewGruppo*;
  2. view, chiave esterna riferita alla tabella *View*.

Di seguito è riportato il diagramma UML con esplicito riferimento a ciascun modello Django implementato (Fig. 42).

Fig. 42:


dashboard.models.clientemodels.Dimensione
dashboard.models.clientemodels.Store
dashboard.models.clientemodels.TipiGruppo
dashboard.models.clientemodels.ViewGruppo
dashboard.models.clientemodels.Metrica
dashboard.models.clientemodels.CategoriaKpi
dashboard.models.clientemodels.Kpi
dashboard.models.clientemodels.ViewGruppo_View
dashboard.models.clientemodels.SourcePlatform
dashboard.models.clientemodels.SourcePlatformAccount
dashboard.models.clientemodels.Cliente
dashboard.models.clientemodels.View
dashboard.models.clientemodels.ViewKpi
dashboard.models.clientemodels.ViewKpiMetricaIntervallo
dashboard.models.clientemodels.Utente
dashboard.models.clientemodels.UtenteCliente
dashboard.models.clientemodels.ViewKpiDimensione
dashboard.models.clientemodels.ViewKpiMetrica
through = 'ViewKpi';
through = 'ViewGruppo_View';
through = 'UtenteCliente';
basestring
unicode
django.contrib.contenttypes.models.ContentType
django.contrib.auth.models.Permission
django.contrib.auth.models.Group
django.contrib.auth.models.User
django.contrib.admin.models.LogEntry
dashboard.models.clientemodels.SeoView
dashboard.models.clientemodels.VisibilityData
dashboard.models.clientemodels.MotoriRicerca
dashboard.models.clientemodels.TempKeywordRequest
dashboard.models.clientemodels.StudioKW
dashboard.models.clientemodels.KeywordData
dashboard.models.clientemodels.CategoriaKW
dashboard.models.clientemodels.Keyword
dashboard.models.clientemodels.CrontabLog
django.contrib.sessions.models.Session
djcelery.models.Failure
djcelery.models.Success
djcelery.models.Schedule
djcelery.models.Site
djcelery.models.OmniQ
djcelery.models.Task
Powered by yFiles

IMPORTAZIONE DB IN PRODUZIONE IN LOCALE

La procedura Fabric auto sviluppata è *rmirror_database_in_local*, che accetta come unico parametro obbligatorio le impostazioni del database in produzione. Il comando effettua una serie di operazioni:

1. recupero delle informazioni di connessione del database in produzione (*get_database_config*) e scaricamento in formato gzip (rdump_database_in_local);
2. estrazione in formato ".sql" dall'archivio (*extract_database_from_gz*);
3. creazione/aggiornamento dei dati nel database locale (*restore_database*).

Di seguito è riportato il codice dei comandi implementati:

```
@task
def get_settings_from_settings_name(settings_name):
      django.settings_module('{SETTINGS_IMPORT_PREFIX}.{settings_
name}'.format(SETTINGS_IMPORT_PREFIX=SETTINGS_IMPORT_PREFIX,
settings_name=settings_name))
      from django.conf import settings
      return settings

def get_database_config(settings_name):
    settings = get_settings_from_settings_name(settings_name)
    return settings.DATABASES["default"]

@task
def rdump_database_to_local(remote_settings, path=TMP_FOLDER,
working_copy=".", dump_name=None):
# settings_name per l'accesso: user e password e quale db
    with lcd(working_copy):
        database = get_database_config(remote_settings)

        if not dump_name:
            dump_name =
"{database[NAME]}.{timestamp}.sql.gz".format(database=database,
timestamp=datetime.now().strftime("%Y%m%d%H%M%S"))
        elif ".sql.gz" not in dump_name:
            dump_name = "{0}.sql.gz".format(dump_name)

        run('mysqldump --user {database[USER]} --
password={database[PASSWORD]} {database[NAME]} | gzip >
/tmp/{dump_name}'.format(database=database, dump_name=dump_name))
        get('/tmp/{dump_name}'.format(dump_name=dump_name),
'{path}/{dump_name}'.format(path=path, dump_name=dump_name))

        return dump_name
```

```
def extract_database_from_gz(dump_name, working_copy="."):
    with lcd(working_copy):
        local(r'bin\7za.exe e {TMP_FOLDER}\{dump_name} -
o{TMP_FOLDER}\ -y'.format(dump_name=dump_name,
TMP_FOLDER=TMP_FOLDER))
        return dump_name.replace(".gz", "")

@task
def restore_database(database_dest, dump_name, working_copy="."):
    with lcd(working_copy):
        local('mysql.exe -uroot -e "DROP DATABASE IF EXISTS
{database_dest};"'.format(database_dest=database_dest))
        local('mysql.exe -uroot -e "CREATE DATABASE IF NOT EXISTS
{database_dest} CHARACTER SET utf8 COLLATE
utf8_general_ci;"'.format(database_dest=database_dest))
        local('mysql.exe -uroot {database_dest} <
{TMP_FOLDER}/{dump_name}'.format(database_dest=database_dest,
dump_name=dump_name, TMP_FOLDER=TMP_FOLDER))


@task
def rmirror_database_in_local(remote_settings,
local_settings=None, path=TMP_FOLDER, working_copy="."):
    """Args: remote_settings, local_settings """
    with lcd(working_copy):
        dump_name_gz = rdump_database_to_local(remote_settings,
path, working_copy)
        dump_name = extract_database_from_gz(dump_name_gz,
working_copy)
        database = get_database_config(remote_settings)

restore_database("{database[NAME]}".format(database=database),
dump_name, working_copy)
```

RIFERIMENTI

[1] https://analytics.google.com

[2] https://w3techs.com/technologies/details/ta-googleanalytics/all/all

[3] https://adwords.google.com/KeywordPlanner

[4] https://authoritylabs.com

[5] https://www.google.com/analytics/360-suite/#?modal_active=none

[6] http://www.tableau.com/

[7] https://aws.amazon.com/it/resources/gartner-2016-mq-learn-more/

[8] https://aws.amazon.com/it/ec2/instance-types/

[9] https://calculator.s3.amazonaws.com/index.html

[10] https://support.hdfgroup.org/HDF5/

[11] http://www.cineca.it/sites/default/files/q5cost_0.pdf

[12] https://www.mongodb.com/it

[13] https://www.gartner.com/doc/reprints?id=1-2PMFPEN&ct=151013

[14] http://pypl.github.io/DB.html

[15] http://www.ingegneria-informatica.unina.it/sites/default/files/elaborati_tesi/2015/01/Elaborato%20Cerbone%20Domenico%20N46000954.pdf

[16] https://speakerd.s3.amazonaws.com/presentations/68d273ab3e034c6787ce90b3faf2376a/Data_Formats_for_Data_Science_-_PyData_EP2016.pdf

[17] http://pypl.github.io/PYPL.html

[18] http://python.org

[19] http://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html

[20] http://airlab.elet.polimi.it/images/archive/8/81/20100922133454!Tesi.pdf

[21] https://pip.pypa.io/

[22] https://virtualenv.pypa.io

[23] http://www.numpy.org/

[24] https://bioinformatics.cineca.it/py/data/01-numpy.pdf

[25] http://pandas.pydata.org/

[26] https://www.djangoproject.com/

[27] http://intermediatepythonista.com/classes-and-objects-ii-descriptors

[28] http://www.django-rest-framework.org/

[29] https://angularjs.org/

[30] https://libscore.com/#angular

[31] https://jquery.com/

[32] https://github.com/mgonto/restangular

[33] https://d3js.org/

[34] http://nvd3.org/

[35] https://lodash.com/

[36] http://instagram-engineering.tumblr.com/post/13649370142/what-powers-instagram-hundreds-of-instances

[37] http://nginx.org/en/

[38] http://gunicorn.org/

[39] http://www.fabfile.org/

[40] http://supervisord.org/

[41] http://www.informit.com/articles/article.aspx?p=2253544&seqNum=4

[42] http://agilemethodology.org/

[43] https://en.wikipedia.org/wiki/PDCA

[44] Jeff Sutherland, *Scrum: The Art of Doing Twice the Work in Half the Time*, 2014

[45] http://www.extremeprogramming.org/

[46] https://www.teamwork.com/

[47] https://slack.com/

[48] https://bitbucket.org/

[49] https://git-scm.com/book/it/v1/Diramazioni-in-Git-Cos-%C3%A8-un-Ramo

[50] https://newrelic.com/

[51] http://www.seleniumhq.org/

[52] https://sentry.io/welcome/

[53] https://cloud.google.com/bigquery/

[54] https://cloud.google.com/files/BigQueryTechnicalWP.pdf

[55] https://spark.apache.org

[56] Prof.ssa Sonia Bergamaschi (dbgroup), Givanni Simonini (dbgroup), Song Zhu (dbgroup), Giuseppe Fiameni (CINECA), Roberta Turra (CINECA), Giorgio Pedrazzi (CINECA) Antonio Macaluso (CINECA), *Course on Data Analytics and Visualization*, 2016, http://dbgroup.unimo.it/big_data_material/03%20-%20%5bSpark-Python%5d%2004%20-%20SparkSQL_intro.pdf

[57] https://www.docker.com/

[58] http://scikit-learn.org/