Università degli Studi di Padova

Facoltà di Matematica

Corso di Laurea in Informatica

Realizzazione di una dashboard per valutare uno strumento di raccomandazione

Laureando

Alberto De Agostini

Relatore

Prof. Mauro Conti

Anno Accademico 2015/2016

Dedicato alla mia famiglia

Indice

1 L'azienda **1**
1.1 Presentazione 1
1.2 Processi aziendali 3
1.3 Tecnologie 4
1.4 Clienti 6
1.5 THRON 8

2 La proposta di stage **13**
2.1 Introduzione 13
2.2 Obiettivi aziendali 15
2.2.1 Obiettivi obbligatori 15
2.2.2 Obiettivi opzionali 15
2.3 Vincoli temporali 16
2.4 Aspettative 16
2.5 Obiettivi personali 16

3 Lo stage **17**
3.1 Piano di lavoro 17
3.2 Analisi 18
3.2.1 Obiettivi delle attività 19
3.3 Tecnologie utilizzate 23
3.3.1 Javascript 23
3.3.2 Snowplow 23
3.3.3 HTML e CSS 24
3.3.4 Scala 24
3.3.5 Apache Spark 25
3.3.6 MongoDB 25
3.3.7 Amazon SNS 25
3.3.8 Elasticsearch 26
3.4 Attività svolte 27
3.4.1 Prima fase 27
3.4.2 Seconda Fase 35
3.4.3 Terza fase 38

3.5 Obiettivi raggiunti 45
3.5.1 Prima fase - Obbligatori 45
3.5.2 Prima fase - Desiderabili 46
3.5.3 Seconda fase - Obbligatori 46
3.5.4 Seconda fase - Desiderabili 46
3.5.5 Terza fase - Obbligatori 47
3.5.6 Terza fase - Desiderabili 48

4 Valutazione retrospettiva **49**
4.1 Obiettivi raggiunti 49
4.2 Competenze acquisite 50

5 Glossario **53**

6 Bibliografia **57**

Elenco delle figure

1.1 workflow aziendale 3
1.2 THRON chat 4
1.3 THRON marketplace 6
1.5 Shareboard in un contenuto THRON 10
1.6 Tag di un contenuto THRON 11
1.4 History di un contenuto THRON 12

2.1 Alcuni dei grafici esposti dalla sezione Intelligence THRON . . 14

3.1 Workflow di Snowplow 24
3.2 Workflow di Amazon SNS 26
3.3 Un widget raccomandativo THRON presente nella home page del sito web aziendale 28
3.4 Workflow di Snowplow personalizzato da THRON 34
3.5 Grafico quantitativo visibile all'interno della dashboard di una applicazione raccomandativa master 41
3.6 Grafico qualitativo visibile all'interno della dashboard di una applicazione raccomandativa slave 42
3.7 Finestra modale aperta al click sul primo contenuto del grafico visto in Figura 3.5 43

L'azienda

Presentazione

THRON è un azienda nata dalla New Vision, una società per azioni italiana fondata nel 2000 da Nicola Meneghello con lo scopo di convincere le aziende ad usare internet come principale mezzo di comunicazione. L'azienda da quando è nata ha sempre avuto un grande interesse per applicazioni web, che non hanno bisogno di installazioni ma sono fruibili direttamente da browser come servizio, questo le rende accessibili da qualunque dispositivo facilitando non poco il lavoro per avere un applicazione multi-piattaforma.
Inizialmente l'azienda sviluppava un software Flash per videoconferenze multiple, successivamente abbandonato questo progetto si sono focalizzati su un prodotto chiamato 4ME il quale è stato integrato nella piattaforma THRON col passare del tempo.
La struttura dell'azienda è gerarchica costituita di quattro livelli: ogni livello è comandato da quello superiore che gestisce l'operato di quelli inferiori. Nel gradino più in alto si trova il consiglio di amministrazione mentre nel livello subito inferiore si trovano diversi settori in cui si divide l'azienda, come:

1. Direzione Tecnica
2. Direzione Marketing
3. Direzione Commerciale
4. Direzione Amministrativa.

Ognuno di questi settori è diviso in team o micro team che costituiscono il terzo e quarto livello. La Direzione Tecnica si divide in 4 team:

1. Engineering
2. Ricerca
3. Formazione

4. Product Specialist

Il team di Engineering che si occupa dello sviluppo del prodotto è divisa in diversi micro team tra cui:

1. Core
2. User Experience
3. Intelligence
4. Marketplace
5. Design

Nel mio periodo di stage sono stato inserito nel team di Intelligence, un team recente che si occupa di raccogliere i dati ed analizzarli per poter dare al cliente una visione degli interessi degli utenti così da poter migliorare le strategie di business e di marketing.

Processi aziendali

THRON non usa un modello di sviluppo standard per il suo prodotto, ma il tutto è molto paragonabile ad un modello agile.
Il progetto è ormai avviato e ben consolidato perciò la sua evoluzione procede attuando piccoli passi incrementali. Questi vengono fatti interagendo in maniera molto forte con i clienti che vengono spesso interpellati per suggerire modifiche e miglioramenti.
Periodicamente vengono fatti degli incontri con clienti ai quali vengono mostrate le evoluzioni, attraverso dei prototipi, per avere dei riscontri sulla qualità e sull'efficacia della soluzione raggiunta.
Dopo questi incontri segue un meeting interno per valutare l'andamento, discutendo sul grado di soddisfacimento del cliente e sulle eventuali proposte mosse da quest'ultimo. Sempre in questi incontri interni vengono presentate anche nuove idee per sviluppi futuri.
Tutte queste nuove idee, miglioramenti, modifiche ed errori vengono scremati dal Project Management$_G$ che le gestisce attraverso un sistema di Ticketing$_G$ aziendale.
Riassumento in un diagramma il flusso delle attività è come in Figura 1.1.

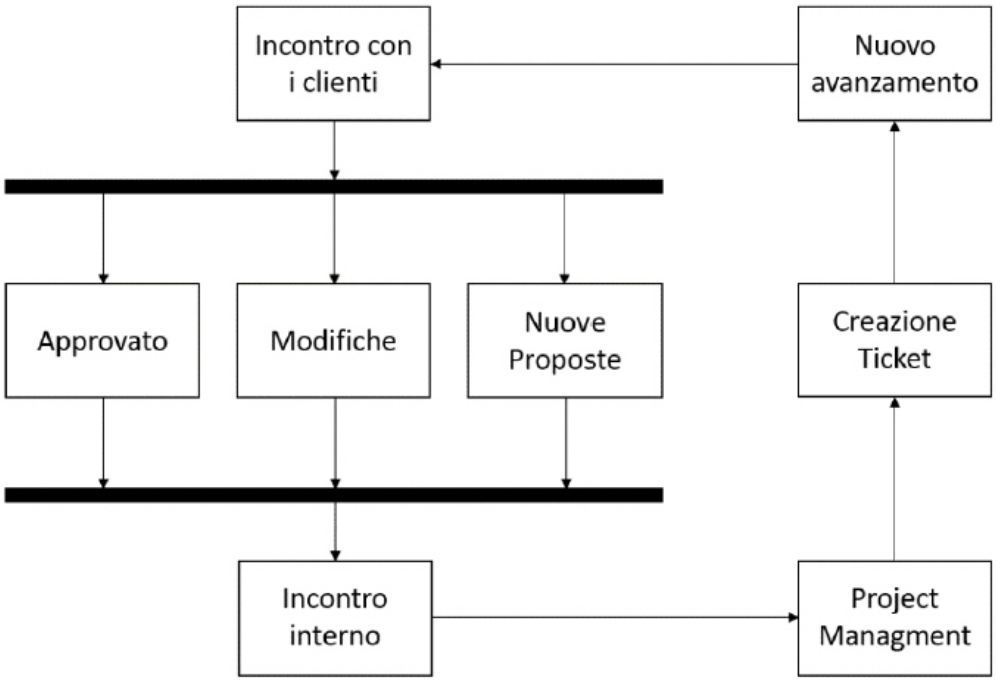

Figura 1.1: workflow aziendale

Tecnologie

Occupandosi l'azienda di molti settori dell'informatica, ed essendo divisa in diversi team, non possiede una suite di strumenti comune per lo sviluppo, perciò ogni team decide a se gli strumenti che ritiene più opportuni per gli obiettivi e le attività che gli vengono assegnati.
In comune ci sono gli strumenti di versionamento, comunicazioni e documentazione.
Per il versionamento l'azienda ha un dominio proprio di gitlab con svariate repositories per ogni progetto.
Per la comunicazione l'azienda utilizza Microsoft Outlook per la posta elettronica, questo perchè è un software che funziona su entrambi i sistemi operativi utilizzati per lo sviluppo (Windows e OS X) e poichè consente una facile gestione di eventi e riunioni su calendario. Ogni membro dell'azienda ha un proprio indirizzo email con la forma *nome.cognome@thron.com*. Per comunicazioni più brevi si utilizza un sistema di messaggistica instantanea interno alla piattaforma THRON come possiamo vedere in Figura 1.2.

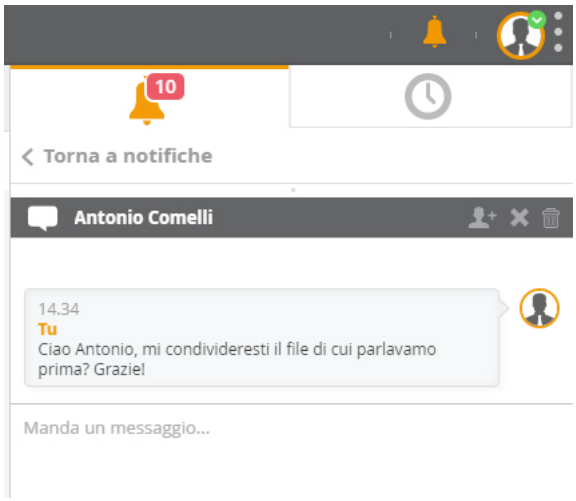

Figura 1.2: THRON chat

Per la documentazione viene utilizzata la suite Office, in particolare Word è utilizzato per la stesura di documenti provvisori e informali, che vengono convertiti in pdf quando il documento viene approvato. Si utilizza anche Power Point per creare presentazioni sia per riunioni interne sia negli incontri con i clienti.
L'ambiente di sviluppo più comune in azienda è Windows, visto il grande supporto per i software utilizzati e per il vasto uso da parte dei clienti, è

presente anche OS X in particolare nei team di Design e User Experience. Viene poi utilizzato Ubuntu per i servizi server per la maggiore stabilità e semplicità in questo ambito. Sono poi presenti e accessibili a tutti delle macchine preconfigurate con sistemi operativi e versioni di browser mirate per testare il comportamento del prodotto su ambienti meno comuni.

Clienti

L'azienda mira ad una vasta ed eterogenea clientela, per questo motivo ha creato un prodotto ampio e facilmente configurabile. In questo modo è il prodotto che si adatta alle esigenze dei diversi clienti. Questo è un punto molto forte, infatti inizialmente il prodotto viene venduto solo con il modulo base che offre le funzionalità standard utili a tutti, poi i vari clienti possono, attraverso la sezione Marketplace, comprare o aggiungere moduli gratuiti che offrono diverse funzionalità rendendo molto flessibile la piattaforma. Possiamo vedere in Figura 1.3 la sezione Marketplace.

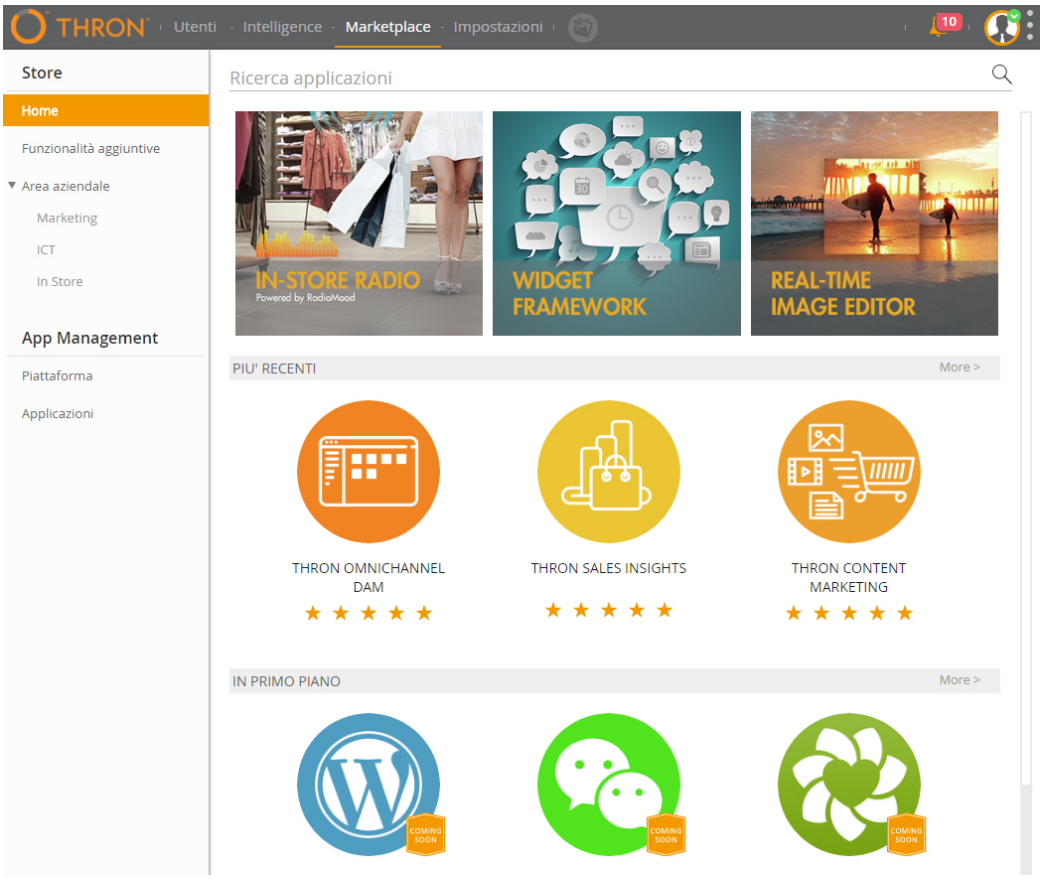

Figura 1.3: THRON marketplace

L'azienda, nel caso queste soluzioni non siano sufficienti, si propone, qual'ora possibile, di creare e confezionare soluzioni personalizzate e mirate per il caso d'uso specifico del cliente.
Le molteplici e diverse richieste da parte dei clienti contribuiscono l'evo-

luzione della piattaforma che deve innovarsi continuamente per soddisfarli. Ognuna di queste evoluzioni conferisce a THRON una completezza sempre maggiore.
In linea generale il client tipo di THRON cerca un modo semplice di gestire una mole di contenuti più o meno grande, di distribuirli su vasta scala sempre mantenendole il controllo.

THRON

L'azienda sviluppa un prodotto omonimo. Questo è pensato per raggruppare in un unico posto tutti gli strumenti per gestire contenuti digitali di ogni tipo.
L'idea che ha portato alla realizzazione di questo prodotto è che i contenuti sono il vero valore, quindi l'organizzazione deve essere slegata dalla piattaforma di distribuzione. Le piattaforme di distribuzione sono in continua evoluzione, ma il contenuto stesso ed il suo valore rimane e rimarrà sempre lo stesso. Pensando al futuro, quando verranno inventate nuove piattaforme (in particolar modo oggetti smart o weareable nel prossimo futuro) THRON sarà pronta a riceverle ottimizzando il contenuto per la fruizione sulla nuova piattaforma con poco sforzo.
La piattaforma supporta molti contenuti come:

- Documenti come pdf, word, excel, HTML, file di testo
- Immagini di ogni tipo comprese immagini vettoriali e bitmap
- Video
- Video e audio on-demand
- Video e audio live

e molti altri.
Per ogni contenuto è possibile scegliere dei permessi, come scegliere che sia visibile solo da alcuni utenti per renderlo privato o visibile solo da un team di interesse, si può dare il permesso di modifica del contenuto o si può garantire il permesso di condividere il contenuto in modo da delegare ad altri la condivizione all'esterno della piattaforma del contenuto.

Per ogni contenuto viene creato uno storico, in questo modo vengono tracciate le azioni fatte su di esso.
Al momento sono tracciate azioni come:

1. Creazione del contenuto (chi lo ha caricato in piattaforma)
2. Modifica dei permessi e quali
3. Visualizzazioni del contenuto, utile per effettuare statistiche di chi li usa, vengono salvate anche le informazioni sul dispositivo utilizzato per la visualizzazione

4. Condivisioni del contenuto

5. Sostituzioni del contenuto, è possibile infatti caricare una nuova versione del file, la vecchia in questo caso viene automaticamente salvata nel caso si volesse tornare ad una versione precedente.

Queste informazioni sono filtrabili per diversi parametri:

1. Data: impostando un arco di tempo predefinito o inserendo data di inizio e di fine

2. Utente o applicazione: vengono visualizzare solo le modifiche fatte da una certa persona o da una speicifica applicazione

3. Tipo: per filtrare attraverso le tag o metadati di ogni contenuto

4. Azione: per filtrare per azione eseguita, ad esempio per vedere solo le visualizzazioni.

Un esempio si può vedere in Figura 1.5 Ogni file viene convertito in diversi formati e dimensioni in modo da ottimizzare la sua fruizione in base al canale di distribuzione finale. Ad esempio un'immagine, una volta caricata, viene copiata e ridimensionata con le varie risoluzioni che si usano maggiormanete nella piattaforma.
A seconda della banda disponibile il sistema controlla che la qualità dell'erogazione dei contenuti sia ottimale ove questo sia possibile. Ad esempio, durante la riproduzione di un video, il sistema controlla la velocità con la quale il client riceve il video e automaticamente aumenta o riduce la qualità. Naturalmente può essere impostata manualmente.

Il prodotto viene infine fornito a siti o servizi esterni da dei connettori, in modo da interfacciarsi e integrarsi senza modifiche alla piattaforma. Per esempio è possibile installare dei connettori per i social network più famosi o per CMS_G come WordPress all'interno di THRON. Questo permette di gestire contemporaneamente un contenuto ovunque esso sia usato. Ad esempio se viene caricata una nuova versione del file questo viene automaticamente modificato ovunque sia stato condiviso. Queste opzioni sono gestite nella sezione Shareboard in THRON come vediamo in Figura 1.6.

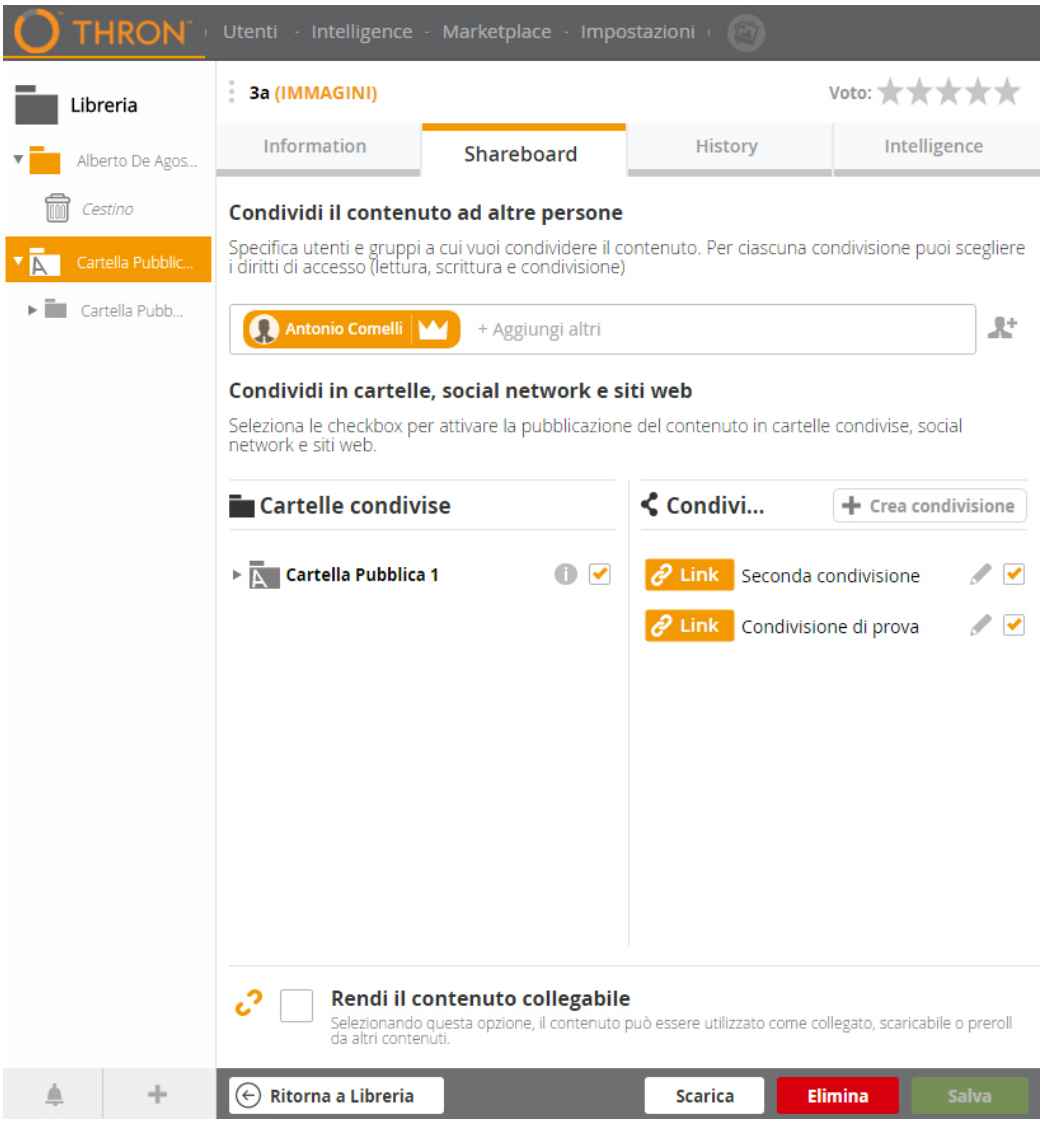

Figura 1.5: Shareboard in un contenuto THRON

Tutti questi contenuti sono raggruppati in un'unica piattaforma che analizza il contenuto e lo arricchisce di informazioni automaticamente con tag$_G$ e metadati$_G$. Questo lavoro è svolto da un motore semantico e analitico che aggiunge queste informazioni sia appena caricato il contenuto sia durante il ciclo di vita di esso. Naturalmente è possibile aggiungere, modificare o eliminare queste informazioni manualmente. Queste, oltre ad arricchire i contenuti di informazioni utili, aiutano anche la loro organizzazione e gestione, soprattutto quando la mole di dati inizia ad essere notevole, poichè si possono ordinare o raggruppare contenuti per tag. Si può vedere la gestione

delle tag in Figura 1.7.

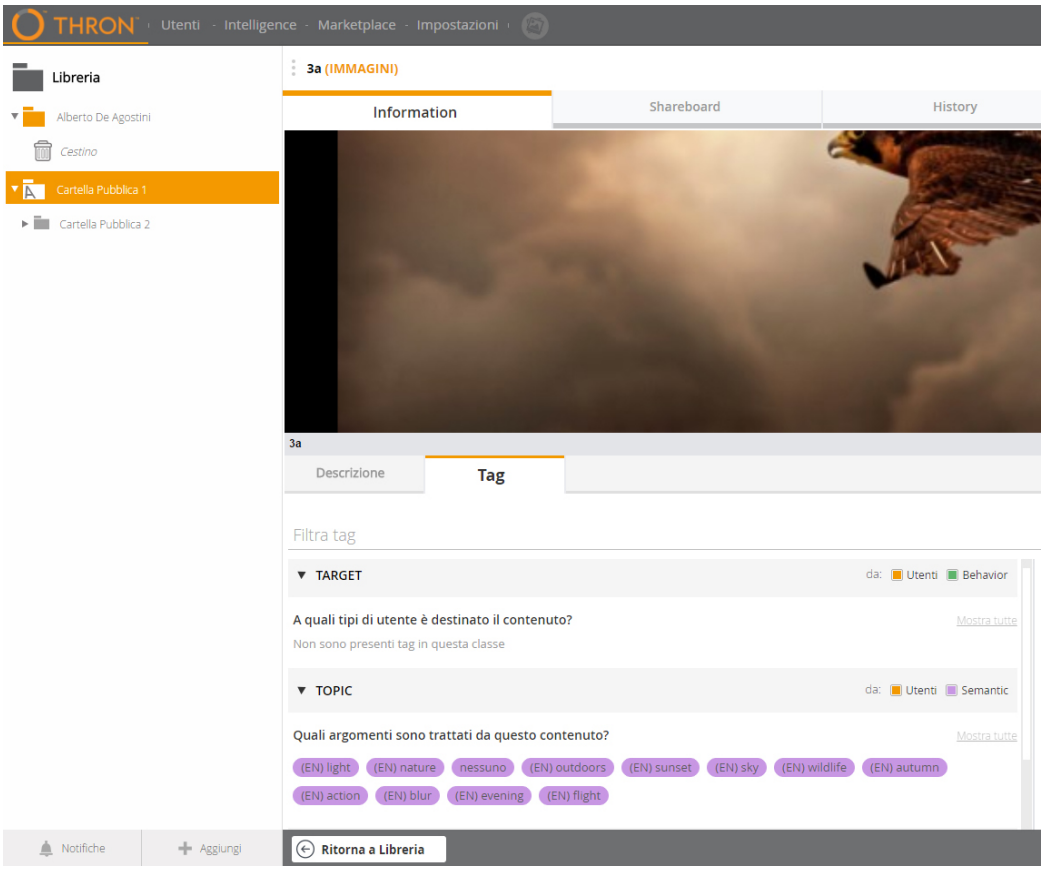

Figura 1.6: Tag di un contenuto THRON

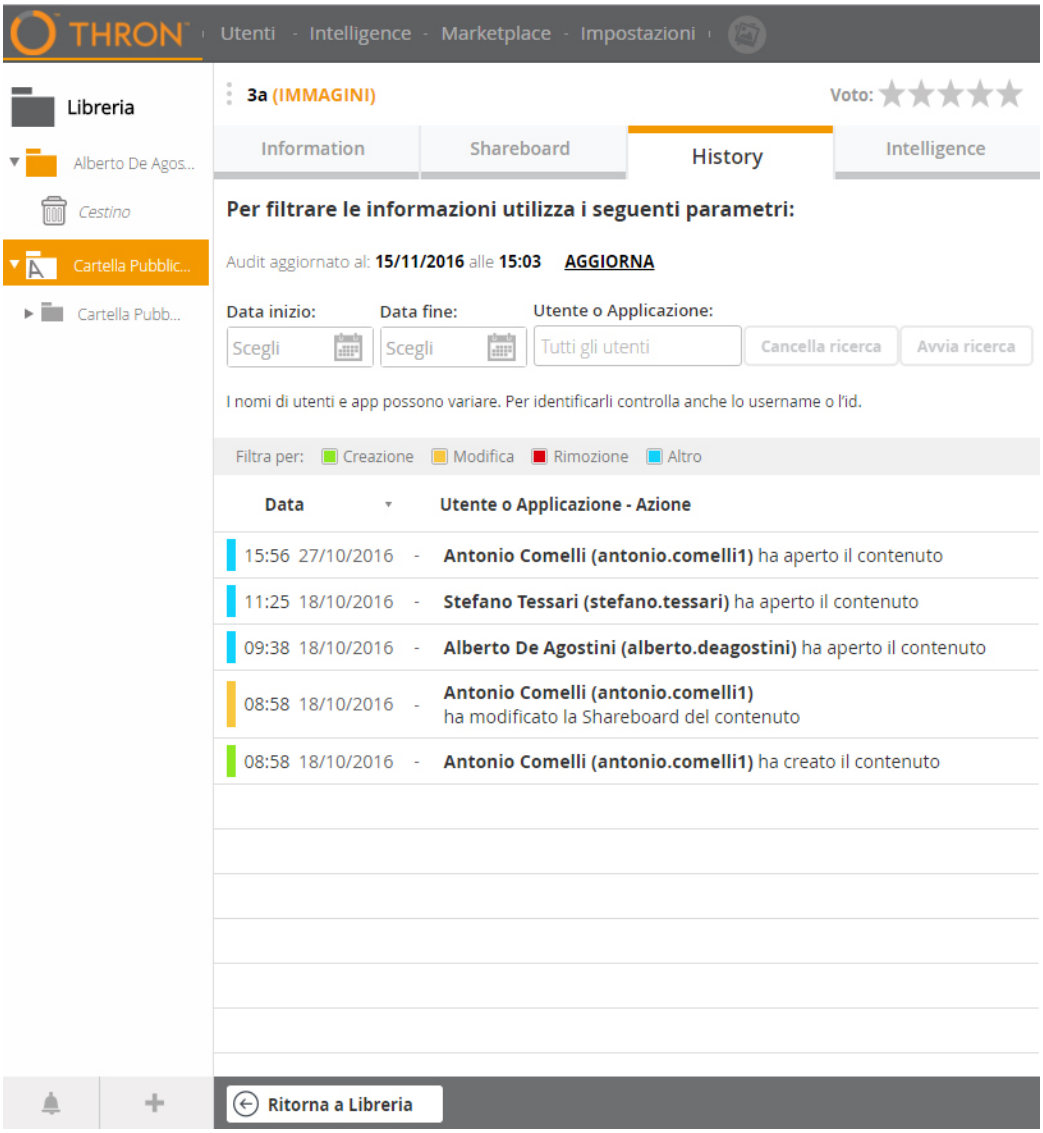

Figura 1.4: History di un contenuto THRON

La proposta di stage

Introduzione

Come già detto in precedenza THRON è una piattaforma che mira a facilitare tutta la gestione dei contenuti. Un punto molto importante per THRON è il poter dare informazioni sui contenuti ai propri clienti per poter migliorare le tecniche di business e di marketing.
Questo non è fatto soltanto aggiungendo tag e metadati ai contenuti ma anche tracciando molteplici informazioni riguardanti la fruizione e le visite sui contenuti stessi sui vari canali di distribuzione con l'uso di una libreria javascript proprietaria.
Le informazioni raccolte vengono poi mostrate sotto forma di grafici di semplice lettura al cliente in apposite sezioni. Un esempio si può vedere in Figura 2.1 Queste informazioni vengono raccolte attraverso una libreria scritta in javascript. Prima dello stage prevedeva la possibilità di tracciare diversi eventi per i video e un solo evento per tutti i tipi di contenuto chiamato **load**. L'evento poteva essere usato con diversi scopi ma nella quasi totalità delle volte era usato per tracciare gli accessi e le visualizzazioni a tutti i contenuti erogati da THRON.
Questo era possibile importando nelle pagine web la libreria dei tracciamenti e aggiungendo qualche riga di codice per inviare le informazioni necessarie. Il tutto era reso automatico nel caso di importazione del player di THRON che al suo interno contiene una libreria dei tracciamenti.

Tra i vari moduli e widget$_G$ che THRON offre ai suoi clienti troviamo il Predictive Content Reccomendation (PCR). Questo è un motore raccomandativo in grado di suggerire, in tempo reale, contenuti in base al profilo degli interessi della singola persona e allo storico delle sue precedenti navigazioni. Grazie alla capacità di THRON di capire gli interessi dell'utente, l'azione del PCR è diversa da quella dei tradizionali sistemi raccomandativi che basano i loro suggerimenti su "cosa hanno fatto o visto utenti simili". THRON invece permette una comunicazione personalizzata in linea con gli specifici

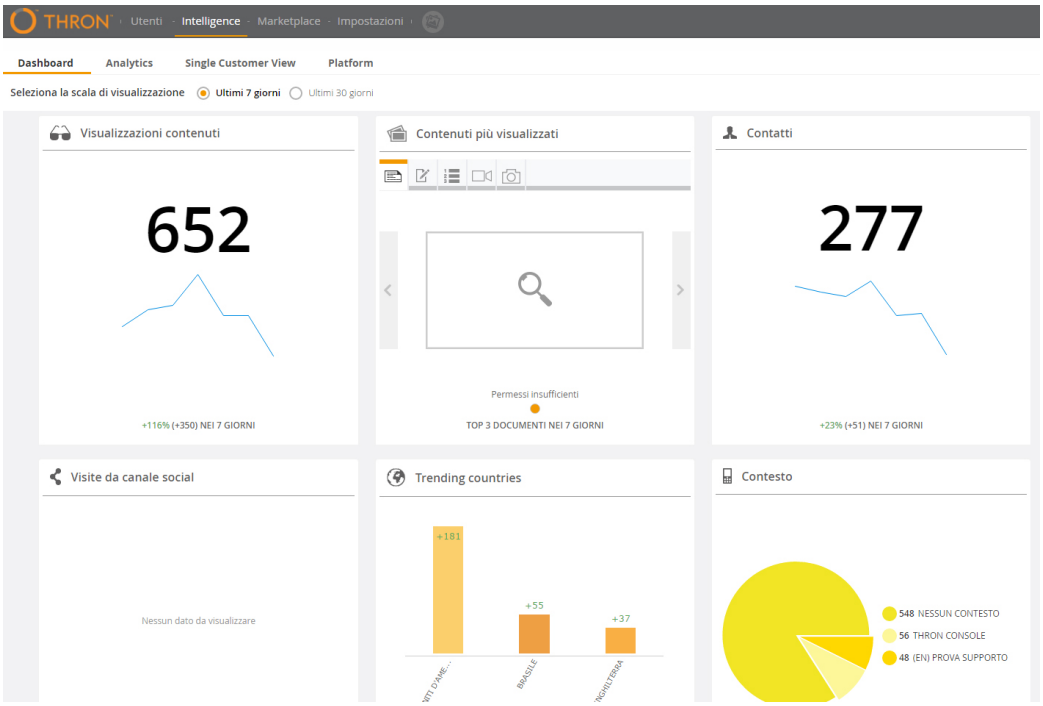

Figura 2.1: Alcuni dei grafici esposti dalla sezione Intelligence THRON

interessi del singolo utente. L'aspetto rivoluzionario del PCR è che, a partire dalla sua capacità di aggregare contenuti da qualsiasi canale, permette di personalizzare la comunicazione su qualsiasi touch point (dal sito web all'app mobile, dall'e-commerce ai social network), aumentando in modo esponenziale il coinvolgimento dell'utente. Il PCR permette di filtrare inoltre contenuti "già visti o fruiti" dalla raccomandazione, supporta il multi-lingua ed estende la sua azione su qualsiasi canale digitale.

Obiettivi aziendali

Il PCR è già disponibile ai clienti ed è composto da due parti: una console di gestione della raccomandazione e una serie di widget "ready to use" non direttamente personalizzabili (al netto di modifiche CSS).
Lo step evolutivo che si vuole ottenere è:

- **Console di monitoring delle prestazioni del PCR**: si tratta di integrare nella console di management già presente una sezione che consente al cliente di verificare le prestazioni di erogazione della struttura raccomandativa; da una prima analisi emergono i seguenti sviluppi plausibili:
  - Diagramma che indica con granularità oraria il numero di richieste per la singola PCR
  - Lista o diagramma che indica lo sforamento del numero di richieste rispetto al modello di business attivato
  - Visualizzazione dello storico delle prestazioni del PCR negli ultimi 30 giorni, con intervallo selezionabile
- **Inserimento di un meccanismo di personalizzazione per i widget**: è necessario implementare una console tramite la quale il cliente scelga il tipo di widget da configurare e ne indichi le personalizzazioni attraverso un interfaccia usabile anche da utenti non tecnici. I widget così configurati potranno essere inseriti all'interno dei progetti dei clienti tramite copia e incolla.

Obiettivi obbligatori

- Creare dei diagrammi che indichino, con granularità oraria, il numero di richieste per la singola applicazione raccomandativa
- Creare una lista o un diagramma che indichi lo sforamento del numero di richieste rispetto al modello di business attivato
- Creare una visualizzazione dello storico delle prestazioni del PCR negli ultimi 30 giorni, con intervallo selezionabile

Obiettivi opzionali

- Aggiungere un meccanismo di personalizzazione per i widget di raccomandazione

Gli obiettivi sono stati poi elaborati e definiti totalmente durante la fase di analisi a cui ho partecipato di cui si può leggere in questa sezione

Vincoli temporali

Per lo stage, l'Università di Padova ha imposto dei vincoli per la durata, che deve essere compresa tra le 300 e le 320 ore. Per questo motivo, prima dell'inizio dello stage, è stato redatto insieme all'azienda un piano di lavoro che comprende 320 ore, contando anche una chiusura aziendale estiva.

Aspettative

L'azienda aveva un forte interesse per questo progetto, questo perché è frutto di diverse richieste da parte dei clienti.
Nello specifico i clienti hanno avanzato richieste per avere informazioni riguardanti quanto le raccomandazioni fossero effettivamente efficaci, l'analisi di questo problema è descritta in questa sezione.
Inoltre questo progetto apre altre evoluzioni specifiche per il motore di raccomandazione che verranno discusse in seguito.

Obiettivi personali

Il mio obiettivo più grande era affrontare uno stage che mi permettesse sia di aprire sbocchi lavorativi all'interno dell'azienda sia di arricchire il più possibile il mio bagaglio personale, affrontando sfide mai viste da me prima e che fossero vere e proprie richieste di un azienda.
A StageIT THRON mi ha fatto subito una buona impressione, proponendomi questo progetto in maniera chiara e facendomi capire che mi avrebbero fatto collaborare con più team coinvolgendomi in tutte le attività legate al progetto così da darmi l'opportunità di ottenere una buona visione del mondo lavorativo.
Inoltre il progetto, nel caso di risultati positivi, sarebbe stato utilizzato come passo evolutivo mettendo la nuova versione in produzione per i clienti.
Questo sommato alle tecnologie da me mai utilizzate o viste solo brevemente nel corso di studi mi ha spinto a scegliere questo progetto.

Lo stage

Piano di lavoro

Con il tutor aziendale era stato redatto un piano di lavoro di 320 ore suddiviso in questo modo:

Prima settimana

- Introduzione all'azienda e alle tecnologie di sviluppo

Seconda settimana

- Fine studio tecnologie e partecipazione alla fase di analisi

Terza settimana

- Continua fase di analisi

Quarta settimana

- Studio e progettazione dell'interfaccia di trasferimento dei dati

Quinta, sesta, settima ed ottava settimana

- Implementazione della console di monitoring e del meccanismo di personalizzazione dei widget

Analisi

La fase di analisi è stata molto importante e più lunga del previsto.
Uno dei motivi che mi hanno spinto a scegliere questo progetto di stage era il fatto che copriva diversi ambiti, non focalizzandosi soltanto, ad esempio, in uno sviluppo backend o frontend, questo ha significato anche il dover interagire con diversi team aziendali effettuando molti brainstorming con componenti di vari team per collegare il tutto.
Per effettuare la dashboard e presentare i dati in maniera elegante ed intuitiva ho collaborato col team di Design e UX per lo sviluppo delle grafiche e l'integrazione in Bacheca$_G$.
Per la raccolta e modellazione dei dati relativi al tracciamento delle singole raccomandazioni di contenuti ho lavorato col team di Intelligence mentre per la raccolta e modellazione dei dati relativi alla quantità di raccomandazioni effettuate dalle varie applicazioni ho lavorato col team Core.
Ho potuto suddividere quindi tutte le attività in tre fasi:

- **fase 1:** collaborazione col team di Intelligence per la raccolta e modellazione dati utili alla creazione dei grafici qualitativi, ovvero quelli per soddisfare le richieste dei clienti relative alla bontà delle raccomandazioni

- **fase 2:** collaborazione col team di core per la raccolta e modellazione dati utili alla creazione dei grafici quantitativi, ovvero quelli per soddisfare le richieste dei clienti relative alla quantità di raccomandazioni fatte ogni ora e la comunicazione in caso di sforamento di richieste;

- **fase 3:** collaborazione col team di design e di UX per lo sviluppo dei grafici e l'integrazione di quest'ultimi all'interno del software Bacheca$_G$.

Obiettivi delle attività

Gli obiettivi iniziali dello stage erano di alto livello e andavano suddivisi in micro obiettivi durante le attività di analisi. Per questo motivo dopo tutte le riunioni e brainstorming fatti coi vari team si sono trovati obiettivi più mirati per la realizzazione dell'intero progetto.

Fase 1

Gli obiettivi di questa fase mirano a sviluppare il necessario per la raccolta ed elaborazione dei dati riguardanti le singole raccomandazioni dei contenuti, utili per soddisfare uno dei macro obiettivi dello stage:

1. Visualizzazione dello storico delle prestazioni del PCR negli ultimi 30 giorni, con intervallo selezionabile

Obbligatori:

- Dovrà essere aggiunta la funzionalità di tracciare impression$_G$ nella libreria dei tracciamenti THRON
- Dovrà essere aggiunta la funzionalità di tracciare click$_G$ nella libreria dei tracciamenti THRON
- Dovrà essere aggiunto un meccanismo per tracciare eventi come click$_G$, load$_G$ e impression$_G$ in automatico nella libreria dei tracciamenti THRON.
- Le modifiche da apportare alla libreria dei tracciamenti dovranno mantenere una retro compatibilità totale
- Tutte le modifiche devono essere supportate nelle seguenti versioni dei browsers:
  - Internet Explorer 9 e successivi
  - Firefox 3.5 e successivi
  - Chrome tutte le versioni
  - Safari OSX 5 e successivi
  - Android 4.1 e successivi: Chrome tutte le versioni
  - IOS 4.x e successivi: Safari 4 e successivi, Chrome tutte le versioni

- Dovrà essere modificato l'enricher per aggiungere il supporto ai nuovi eventi
- L'enricher dovrà salvare tutti gli eventi in indici diversi raggruppati per tipo di evento
- Aggiungere al modulo Reports il supporto ai nuovi eventi
- Far si che gli altri moduli di Intelligence ignorino questi nuovi eventi

Desiderabili:

- Il meccanismo contenente la logica di automazione degli eventi dovrà essere aperto ad estensioni per futuri eventi

Fase 2

Gli obiettivi di questa fase mirano a sviluppare il necessario per la raccolta ed elaborazione dei dati riguardanti la quantità di raccomandazioni, utili per soddisfare 2 dei macro obiettivi dello stage:

1. Diagramma che indica con granularità oraria il numero di richieste per la singola PCR
2. Lista o diagramma che indica lo sforamento del numero di richieste rispetto al modello di business attivato

Obbligatori:

- Creare un servizio che interroghi Elasticsearch di monitoring, ricavi i dati degli accessi alle raccomandazioni e li salvi in un database
  - Questo servizio deve salvare gli accessi con granularità oraria
  - Questo servizio deve salvare gli accessi con granularità giornaliera
  - Questo servizio deve salvare gli accessi con granularità mensile
- Sviluppare dei web services interrogabili dall'esterno che permettano di interrogare il database
  - Questi web services devono prevedere politiche di accesso e un sistema di permessi uguali agli altri web services THRON

Desiderabili:

- Sviluppare un servizio che ripulisca i dati, con granularità oraria e giornaliera, più vecchi di due anni

Fase 3

Gli obiettivi di questa fase mirano a sviluppare una dashboard con all'interno il necessario per la presentazione dei dati raccolti ed elaborati nelle fasi precedenti.
Sono quindi utili per soddisfare tutti gli obiettivi dello stage poiché tutti i grafici utilizzano i dati raccolti in precedenza.
Obbligatori: Tutti i seguenti obiettivi devono essere implementati all'interno di una dashboard da integrare nell'applicazione Bacheca$_G$ di THRON.

- Realizzare un grafico che indichi il numero di raccomandazioni fatte da un applicazione raccomandativa master (grafico quantitativo)
  - Questo grafico deve permettere la scelta di diverse granularità, in particolare oraria, giornaliera e mensile
  - Questo grafico deve permettere la scelta di una data di inizio e di fine per filtrare i dati
  - Questo grafico deve poter mostrare, con possibilità di filtraggio, il numero di raccomandazioni delle singole applicazioni raccomandative slave
- Per ogni applicazione slave devono essere realizzati altri grafici per analizzare la bontà delle raccomandazioni (grafici qualitativi)
  - Un grafico che mostri i cinquanta contenuti raccomandati visti più volte (ovvero con più impression$_G$) con la relativa quantità di click sul contenuto
  - Un grafico che mostri gli argomenti relativi ai contenuti più visti
  - Un grafico che mostri il profilo dei contatti associato ai contenuti più visti
  - Al click su un contenuto di questo grafico si apra un popup che contenga
    - Una descrizione del contenuto
    - Il numero di impression$_G$
    - Il numero di click$_G$
    - I tag del contenuto
    - Le tipologie degli utenti che hanno interagito col contenuto
    - Gli argomenti più trattati da queste tipologie di utenti
- Dare la possibilità di filtrare per periodo qualunque

- Dare la possibilità di cercare specifici contenuti
- Dare la possibilità di ordinare per più visti o più cliccati

Opzionali:

- Sviluppare una sezione in cui vengono dati consigli sulle raccomandazioni, ad esempio consigliare quali contenuti si consiglia di togliere dalle possibili raccomandazioni
- Sviluppare un meccanismo per la personalizzazione dei widget di raccomandazione

Tecnologie utilizzate

Javascript

Javascript è un linguaggio di scripting orientato agli oggetti e agli eventi. Principalmente è utilizzato per la programmazione lato Client negli ambienti Web per aggiungere funzionalità, effetti dinamici o interattività alle pagine o applicazioni web.
Da qualche tempo è però utilizzato in molti altri ambiti come la programmazione di applicazioni mobile tramite frameworks come ionic o appcelerator.
Essendo Thron una piattaforma web, Javascript è un linguaggio pressoché obbligatorio da imparare per lavorare in azienda.
Nel mio stage l'ho utilizzato largamente per apportare le modifiche alla libreria dei tracciamenti
In più l'ho utilizzato anche per aggiungere il tracciamento automatico nei widget di raccomandazione, essendo anche quest'ultimi scritti in javascript.
Inoltre ho utilizzato javascript anche per l'esposizione dei dati, ovvero per la creazione della console in cui sono esposti dei grafici come spiegato nei capitoli a seguire.

Snowplow

Snowplow è un complesso sistema basato su pipelines$_G$ per effettuare il tracciamento di eventi, per la raccolta di quest'ultimi e per la loro elaborazione. Fornisce librerie scritte in molti linguaggi di programmazione da inserire nel proprio applicativo per effettuare tracciamenti personalizzati. Nel mio caso ho usato quella scritta in Javascript.
Due sono i punti forti di Snowplow:

1. I dati relativi ai tracciamenti restano di proprietà dell'azienda

2. I tracciamenti sono completamente personalizzabili

Questi aspetti lo rendono molto diverso da altri concorrenti come Google Analytics. Il rovescio della medaglia è che è necessario molto più tempo per creare tutta la pipeline per supportare tutto il workflow di Snowplow, che è sintetizzato in Figura 3.1.

Figura 3.1: Workflow di Snowplow

I trackers rappresentano le librerie da aggiungere nei client, queste attraverso delle chiamate HTTP a dei webhook inviano i dati in un collector. Questo è uno Scala Stream Collector, ovvero un software Scala che riceve gli eventi grezzi di snowplow attraverso chiamate HTTP, le serializza e scrive i dati in Amazon S3. L'enricher è un software che carica i dati da S3 e li arricchisce con ulteriori informazioni.
Snowplow fornisce una versione software di base per ogni blocco, tuttavia sono necessari cambiamenti o diverse implementazioni per necessità personalizzate come nel caso THRON.
Per rendere possibile il tracciamento di dati eterogenei e personalizzabili Snowplow adotta un sistema di schemi. L'idea è creare degli schemi in cui si descrive come può essere un evento tracciato lato client, in questo modo il server che riceve questi eventi può validarli usando questi schemi.

HTML e CSS

Naturalmente per la creazione dei grafici e l'integrazione in Bacheca$_G$ ho utilizzato anche questi linguaggi.
Ho creato anche diverse pagine di demo e di test per il parser$_G$ creato per effettuare i tracciamenti in automatico.

Scala

Scala è un linguaggio di programmazione orientato alla programmazione funzionale con una forte tipizzazione.
Il codice Scala viene compilato in bytecode Java così l'esecuzione avviene sulla Java Virtual Machine.
In questo modo si possono utilizzare libreria Java all'interno di Scala rendendolo un ambiente molto vasto e pieno di librerie già pronte da riutilizzare.
Una peculiarità di Scala è data dal fatto che, come dice il nome stesso, è

orientato alla scalabilità orizzontale, questo lo rende un linguaggio di programmazione in forte ascesa negli ultimi anni, soprattutto per aziende che trattano BigData o una certa elasticità di richieste per i loro servizi.

Apache Spark

Spark è un framework per l'elaborazione dati su cluster. Supporta diversi linguaggi di programmazione quali Java, Scala e Python e nel mio caso ho utilizzato Scala.
L'obiettivo di Spark è eseguire grandi quantità di elaborazioni dati su cluster il più velocemente possibile. Costruito in risposta alle limitazioni del paradigma computazionale MapReduce che veniva utilizzato prima può vantarsi di essere cento volte più veloce di Hadoop MapReduce quando eseguito in memoria.
L'ho utilizzato per tutta la pipeline di elaborazione dati relativa ai tracciamenti effettuati con Snowplow. In particolare, riferendosi al workflow di Snowplow visto in Figura 3.1 , il blocco di Enrichment è sviluppato con Spark.

MongoDB

MongoDB è un database orientato ai documenti gratuito e open source. È considerato un NoSQL e usa dei documenti stile JSON con schema.
È un database molto utilizzato e presenta tantissime utilità e driver scritti in ogni linguaggio di programmazione. La sua flessibilità, scalabilità e le buone performance fanno di MongoDB uno dei database più utilizzati.
Nel progetto è stato utilizzato per il salvataggio dei dati relativi al numero di raccomandazioni.

Amazon SNS

Amazon SNS è un servizio di notifiche push. Permette di inviare messaggi individuali o collettivi ad un numero elevato di destinatari. I destinatari possono essere dispositivi mobili, destinatari di posta o anche altri servizi distribuiti.
Il workflow di SNS è come descritto in Figura 3.2. Il publisher può essere una persona o un programma che scatena un evento, l'SNS topic è una coda a cui software può effettuare sottoscrizione per essere notificati qual'ora ci sia un evento.

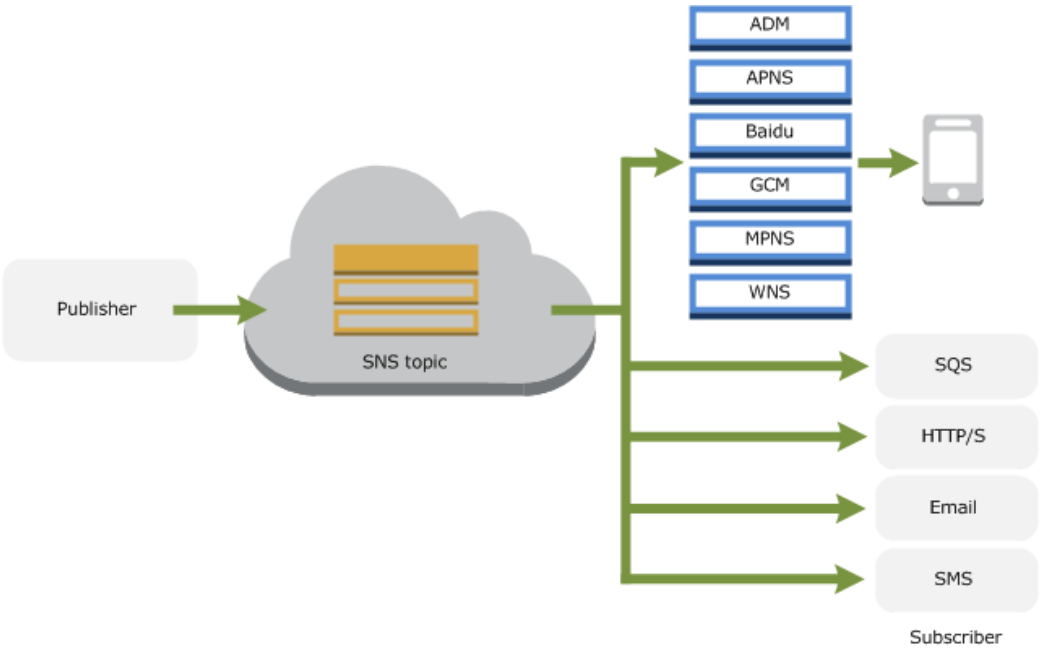

Figura 3.2: Workflow di Amazon SNS

Utilizzato per creare dei servizi lato server che seguano uno scheduling costante.

Elasticsearch

Elasticsearch è un motore di ricerca basato su Lucene$_G$. Consiste in un motore di ricerca distribuito interrogabile tramite richieste HTTP. All'interno i dati vengono salvati sotto forma di documenti JSON. Le prestazioni sono così elevate da essere considerato real-time, per questo ha ottenuto sempre maggiore successo e al momento è utilizzato nella maggior parte dei progetti in cui si deve interrogare grandissime quantità di dati distribuibili.
Nel progetto mi sono interfacciato con diverse istanze di Elasticsearch poichè l'azienda ne fa largo uso, in particolare il team di Intelligence che ha a che fare con BigData.

Attività svolte

Prima fase

Analisi tracciamenti

Per prima cosa si è scelto di analizzare come raccogliere i dati necessari per poter capire e fornire al cliente delle metriche e valutazioni di efficienza dello strumento PCR. Infatti i clienti tra le richieste volevano sapere quanto efficace il PCR fosse, questo però è un concetto non basilare e non definito dai clienti stessi perciò molto tempo dell'analisi è stato impiegato a pensare come valutare questo.
Ho collaborato con il team di Intelligence e con uno data scientist$_G$ per cercare di definire questo concetto. Sono state vagliate diverse proposte interne e fatte diverse ricerche in internet, alla fine si è scelto l'approccio che conciliasse una buona precisione sull'efficacia dello strumento dovendo però rientrare in tempi non molto lunghi per i vincoli di stage.
La definizione risultate è la seguente:
per ogni contenuto devono essere tracciate almeno tre azioni:

1. Contenuto proposto
2. Contenuto proposto e visto dall'utente
3. Contenuto proposto, visto e l'utente ha espresso un interesse esplicito

D'ora in poi useremo terminologie specifiche per questi eventi, in particolare

1. Load: contenuto proposto
2. Impression: contenuto proposto e visto dall'utente
3. Interact o click: contenuto proposto, visto e l'utente ha espresso un interesse eplicito, ad esempio cliccandoci

Tramite questi eventi l'azienda è in grado di fornire al cliente varie informazioni tra cui:

- I contenuti più proposti
- I contenuti più visti
- I contenuti più cliccati

Mentre è facile capire che i contenuti più proposti siano i contenuti caricati più volte dai widget di raccomandazione (ovvero contenuti caricati nel widget nella pagina web), è importante capire la differenza tra visto e cliccato. Per contenuto visto (impression) si intende un contenuto proposto dal PCR e che è entrato nella viewport$_G$ dell'utente finale (ma che non necessariamente l'utente ci ha interagito). Per contenuto cliccato si intende un contenuto visto dall'utente in più su cui quest'ultimo ha fatto un click per aprirlo o per interagirci.
Infatti i widget di raccomandazione contengono una lista di contenuti con una thumbnail$_G$ e una breve descrizione, al click su un contenuto si è portati nella pagina del contenuto o in un altra pagina web a seconda di come è impostato il widget. Un esempio di un widget raccomandativo lo possiamo vedere in Figura 3.3

Figura 3.3: Un widget raccomandativo THRON presente nella home page del sito web aziendale

Durante l'analisi si è capito che l'azione di caricare il contenuto dal widget (evento load) non portava necessariamente informazioni molto utili riguardanti le raccomandazioni, poiché se un contenuto non è neanche entrato nella viewport dell'utente si può semplicemente ignorare. Abbiamo così deciso di tracciare questo evento per avere questa informazione che può essere utile in futuro, ma per ora ci concentreremo più sugli altri due eventi.
Abbiamo infatti, dopo brainstorming e ricerche in internet, trovato una metrica che dasse un indicazione soddisfacente dell'efficacia di un contenuto, questa metrica si chiama **click through rate**. In pratica è un valore calcolato dal rapporto delle interazioni con il contenuto con il numero di viste (click diviso impression). Questo indice può assumere valori compresi tra zero e uno, in particolare un valore vicino all'unità sta ad indicare che il contenuto ha subito molte interazioni rispetto alle visite ricevute. Questo fatto indicherebbe che il contenuto raccomandato è interessante per quasi tutti gli utenti a cui è stato proposto indicando un ottimo lavoro da parte del motore di raccomandazione. Valori invece molto bassi di questo indice significherebbero che il contenuto è stato aperto poche volte rispetto al numero di visite, suggerendoci che il contenuto ha provocato scarso interesse da parte degli utenti a cui è stato proposto.

Si è deciso quindi che questo fosse uno dei valori, per ogni contenuto, da presentare al cliente.
Abbiamo anche deciso di eliminare casi particolari come gli elementi con un numero di visite al di sotto di una certa soglia. Questo perché uno scarso numero di visite e pochi click avrebbe comunque un click through rate più alto di contenuti magari più significativi. Per esempio un contenuto con 3 click e 4 impression avrebbe un click through rate di 0.75, indice molto alto, mentre un contenuto con 1000 impression e 500 click avrebbe un click through rate di 0.5, il che porterebbe a pensare che il contenuto sia molto meno interessante ma è probabile che in realtà sia il contrario perché con numeri molto bassi è facile avere un rapporto alto e non veritiero.
Tutte queste considerazioni valgono non solo per contenuti caricati dai widgets di raccomandazione ma per qualunque contenuto THRON messo in pagina. Questo è stato motivo di ulteriore stimolo poiché erano funzionalità aggiunte per ora sullo specifico caso, ma con la possibilità in futuro di essere estese ad altri progetti dell'azienda.

Progettazione e sviluppo prima fase

Raccolta degli eventi - libreria dei tracciamenti

Per raccogliere le informazioni necessarie THRON usa una libreria javascript che espone dei metodi per tracciare eventi. Questa libreria deve essere inclusa nelle pagine web o applicazioni e aggiunta in pagina la logica per tracciare queste informazioni. Un esempio banale: se si volesse acquisire l'informazione di quante volte viene aperta la scheda di un contenuto in un sito di e-commerce basterebbe lanciare un evento load al caricamento della pagina relativa dell'oggetto da vendere sul contenuto di tipo immagine relativo all'oggetto.
La libreria dei tracciamenti THRON supportava diversi eventi da loro già tracciati, tuttavia non disponeva di metodi per tracciare impression e click perciò era necessario aggiungere queste funzionalità.
Per fare questo ho dovuto studiare come era implementata la libreria e familiarizzare con diverse tecnologie mai utilizzate da me prima tra cui:

- **grunt:** è un javascript task runner, ovvero una libreria per automatizzare azioni che si eseguono spesso. Nel mio caso l'ho utilizzato per automatizzare la compilazione, la minimificazione e l'esecuzione di batterie di test

- **Lodash:** è un evoluzione di **underscore.js**, si tratta di una libreria che contiene moltissime funzioni di utilità con il massimo delle performance

- **ajax:** per effettuare chiamate http

e molte altre di minore importanza, alcune delle quali per aggiungere il supporto a funzionalità altrimenti non supportate da alcuni browser, ad esempio per utilizzare le promises$_G$ su Internet Explorer 9.
L'inizio è stato difficile, dovevo capire perfettamente come funzionasse la libreria dei tracciamenti e poi capire come funzionasse e come fosse il workflow di Snowplow che è utilizzato all'interno della libreria per inviare i dati. La libreria infatti è molto complessa e non avevo mai dovuto studiare un modulo così grande durante i miei studi universitari.
Fortunatamente la libreria era scritta molto bene, documentata e presentava delle pagine di demo che aiutavano molto a capire e a provare tutte le funzionalità. L'aggiunta dei nuovi eventi impression e click è stata semplice infine. Utilizzando l'ereditarietà prototipata di javascript ho creato un oggetto che ereditasse tutte le funzionalità già presenti e ho aggiunto due nuovi metodi per esporre al client la possibilità di tracciare questi eventi. Vista la natura di snowplow è stato necessario anche aggiungere degli schemi perché i nuovi tracciamenti avevano bisogno di inviare anche nuove informazioni come l'identificativo della applicazione di raccomandazione.
Questo era sufficiente per gli obiettivi riguardanti l'aggiunta degli eventi nella libreria dei tracciamenti, tuttavia per effettuare il tracciamento vero e proprio era ancora necessario che qualcuno scrivesse qualche riga di codice in tutte le pagine dove era necessario tracciare questo nuovo tipo di evento. Inoltre il codice per capire quando un contenuto entri nella viewport$_G$ di un utente non è affatto banale e non era accettabile che in ogni pagina si dovesse aggiungere questo tipo di informazioni.
Era quindi necessario creare un sistema che tracciasse in automatico, secondo logiche prefissate, contenuti THRON in pagine web senza l'ausilio di aggiunta di codice.
Il problema è molto ampio e non banale, tuttavia questo stage rappresentava un buon punto per iniziare.
I widget raccomandativi, come visto in precedenza, aggiungono in pagina una lista di immagini con descrizione. Queste immagini, chiamate thumbnail$_G$, rappresentano effettivamente il contenuto stesso quindi si è scelto di partire con un software che tracciasse in automatico eventi sulle immagini, lasciandolo aperto a future estensioni per tutti gli altri tipi di contenuto che, spesso, useranno le stesse logiche applicate per le immagini.
Ho pensato quindi di creare un modulo all'interno della libreria dei tracciamenti che usasse le funzionalità esposte dalla libreria stessa.
L'idea che ho avuto è stata creare un *parser*, ovvero un oggetto che, al caricamento della pagina, o tramite invocazione javascript, analizzasse la pagina

web e cercasse tutti gli elementi possibili da tracciare.
Due problemi erano presenti: come far capire al parser quali erano gli elementi tracciabili e come recuperare le informazioni per il tracciamento, infatti la libreria dei tracciamenti espone un oggetto chiamato *tracker* su cui invocare gli eventi da tracciare. Quest'ultimo necessità di informazioni come l'identificativo del contenuto, il codice servizio$_G$, il tipo del contenuto e altro ancora.
Il mio tutor mi ha lasciato libera scelta per il primo problema e mi ha indirizzato invece su una facile soluzione per il secondo.
Mi ha mostrato che le thumbnail$_G$ usate per rappresentare i contenti nelle pagine web avevano un url costruito secondo un pattern ben definito, ovvero:

https:\\<codice_servizio>-view.thron.com/api/xcontents/resources/ delivery/getThumbnail/<codice_servizio>/<dimensione>/ <id_contenuto>.<formato>

Da questi url è facile estrarre il codice servizio, l'id del contenuto e il tipo di contenuto è sempre un immagine.
Rimaneva da trovare una soluzione per far capire al parser quali elementi in pagina erano da tracciare e quali azioni tracciare su di essi. Infatti nel caso di questo progetto era necessario tracciare gli eventi load, click e impression, tuttavia il modulo aggiuntivo doveva essere progettato pensando a sviluppi futuri perciò doveva dare la possibilità ai clienti di scegliere quali azioni tracciare su quali contenuti, non necessariamente tutti i contenuti THRON inclusi in pagina.
Ho scelto di risolvere entrambi questi problemi attraverso l'uso di classi CSS. L'idea è quella di aggiungere una classe, personalizzabile, per far capire al parser quali elementi devono essere analizzati e quali azioni tracciare in automatico su di essi.
Ho analizzato il problema col tutor e ha accettato la mia idea per la semplicità e perché non rendeva necessaria nessuna modifica a servizi come quello che genera le thumbnail$_G$ che avrebbe causato ritardi nella realizzazione.
Ho così iniziato a sviluppare questo modulo nella libreria, mantenendo una struttura ad oggetti per lasciare aperte future estensioni per il supporto di nuovi tipi di contenuti o di nuovi eventi. Su suggerimento del tutor ho aggiunto dei controlli per filtrare eventi non possibili su alcuni tipi di contenuto, come tracciare eventi seek$_G$ su immagini e per limitare gli errori possibili di battitura sulle classi.
L'idea che avevo avuto inizialmente era di sviluppare questo parser utilizzando tutte le funzionalità esposte dal framework jQuery, questo infatti mi avrebbe aiutato molto per la ricerca degli elementi, per il filtraggio e per avere un supporto multi browser assicurato. Tuttavia mi è stata sconsigliata perché è una libreria molto grande e avrebbe appesantito l'intera libreria.

Ho così sviluppato tutto il modulo in puro javascript senza l'ausilio di altri framework eccetto quelli che già erano nella libreria. A lavoro finito devo dire che il tutor aveva pienamente ragione poiché senza l'uso di jQuery la libreria occupa circa **100KB** mentre con l'uso di jQuery circa **180KB**.
Il supporto ad alcuni browser, in particolare Internet Explorer 9, mi ha dato qualche problema che ho dovuto risolvere con dei fallback o aggiungendo dei piccoli polyfill$_G$ che potranno essere utili anche in futuro.
Ho reso poi il parser parametrico, utile in particolar modo la possibilità di scegliere di analizzare solo pezzi della pagina web identificabili tramite HTML id.
Ho creato delle pagine di demo apposite piene di immagini con tutti i casi limite a cui ho pensato per provare il modulo e per rendere anche più facile la comprensione per chi dovrà mantenerlo, mi sono ispirato infatti alle pagine di demo già presenti nella libreria dei tracciamenti che mi avevano aiutato moltissimo nella comprensione del software.
Ho sviluppato il parser per far si che non analizzi soltanto il codice HTML nel momento della sua istanziazione ma attraverso l'uso di mutation observers$_G$ continua a essere in ascolto per cambiamenti nel DOM$_G$. Se ne avvengono nella parte di HTML che è stato istruito ad analizzare controlla se questi cambiamenti contengono le classi di interesse o se sono stati aggiunti nuovi elementi tracciabili, questo ultimo fattore rende tutto il tracciamento automatico senza aggiunta di codice in pagina come richiesto, eccetto una riga per istanziare il parser che è inevitabile.
Ho poi aggiunto le logiche per automatizzare i tracciamenti. Per l'evento load la logica era molto semplice: load sta a significare che il contenuto è stato caricato in pagina, perciò il parser non appena trova un contenuto che nella classe gli dice di tracciare l'evento load lancia l'evento.
Per il click è similmente facile, basta aggiungere un listener all'elemento HTML che al click dell'utente fa partire l'evento.
Un altra storia è per l'evento impression: questo evento deve essere lanciato quando l'immagine in pagina entra nella viewport$_G$ dell'utente. Ho cercato molto online, sono presenti diverse librerie che cercano di risolvere il problema ma tutte hanno diversi problemi quindi ho sviluppato io la logica.
L'idea di base è aggiungere dei listener allo scroll della pagina e fare dei controlli sulla posizione degli elementi rispetto alla pagina, il tutto era già fatto dagli script che si possono trovare con una piccola ricerca online ma il problema diventa più complesso quando si iniziano a considerare altri aspetti come lo scrolling orizzontale e il fatto che un elemento può essere all'interno di un DIV$_G$ anch'esso scrollabile.
Ho quindi creato una funzione che analizza la posizione di un elemento rispetto alla pagina e rispetto al primo e al secondo elemento HTML scrollabile

tra i suoi ascendants$_G$ nel caso ci siano.
Ho scelto di non andare oltre al secondo livello di parentela per due motivi:

- È improbabile che ci sia un immagine all'interno di tre livelli di scroll in una pagina
- I calcoli necessari iniziavano a diventare molti e potrebbero impattare negativamente le performance di una pagina web

I primi test non facevano ben sperare. Ho effettuato i primi test in una delle mie pagine appositamente create che contenevano centinaia di immagini con tutti i tipi di scroll. Avevo aggiunto anche dei widget THRON che forniscono una serie di immagini automaticamente ridimensionate in una sezione HTML e che, allo scroll, ne carica altre tramite chiamate Ajax.
Ho notato subito dei piccoli rallentamenti nello scroll della pagina. Ho pensato che fosse perché c'erano troppe immagini, tuttavia il numero di queste, in un caso d'uso reale, non ha virtualmente un limite quindi questo sistema di tracciamento automatico doveva funzionare anche con centinaia di immagini caricare.
Dopo un'attività di attento debug ho visto che il browser, allo scroll del body o di un DIV$_G$, va a innescare la callback impostata nello scroll listener molto volte. Ho così implementato un rudimentale sistema simile ad un sistema di locking per risorse per far si che la callback venisse eseguita una sola volta ed ho riprovato la navigazione in pagina.
Tutta la navigazione risultava essere completamente senza rallentamenti ora, in più il tracciamento automatico funzionava come sperato.
Il tutor ha approvato il lavoro fatto e il parser è stato anche messo in alcune pagine del sito di THRON per testarlo in un caso d'uso vero dandomi molta soddisfazione.
Per aggiungere il tracciamento automatico nei widgets di raccomandazione mi è bastato aggiungere nel codice del widget stesso le istruzioni per istanziare un parser e per far partire il parsing della sezione HTML in cui il widget è contenuto, questo infatti è associato ad un id che il widget conosce.
L'analisi e la progettazione di queste attività sono state più lunghe del previsto, si è deciso quindi di cercare di risparmiare tempo nelle prossime attività qual'ora ci fosse l'opportunità, altrimenti di tralasciare qualcuno degli obiettivi non obbligatori come lo strumento di personalizzazione dei widget.

Elaborazione dei dati raccolti - Spark
Come già detto parlando di Snowplow prevede che i dati vengano raccolti nel collector, qui si troveranno tutti i dati dei tracciamenti grezzi, così come sono inviati dai client. Per renderli presentabili all'utente finale hanno bisogno di

alcune elaborazioni che sono fatte da degli enricher.
THRON ne ha sviluppato diversi per i propri bisogni, quattro in particolare:

- Un enricher comune che prende i dati dal collector e va ad aggiungere tutte quelle informazioni che sono necessarie per tutti i dati, indipendentemente da dove essi debbano essere presentati. Questo infatti aggiunge informazioni relativi a chi ha scatenato gli eventi. Questo modulo infatti associa all'identificativo utente tracciato, un identificativo utente THRON tramite il merge con i dati di un database apposito. Questo processo fa si che si identifichi univocamente l'utente che ha effettuato una certa azione

- Un enricher per il recommendation che legge i dati in uscita del primo enricher e li elabora secondo algoritmi scritti da THRON per produrre in output dei json che servono per la raccomandazione di contenuti

- Un enricher per il behaviour che serve ad analizzare le tag dei contenuti più frequentemente accedute da parte degli utenti per associarle ad essi

- Un enricher per i reports che legge i dati arricchiti con le informazioni dei contatti e va ad aggiungere una serie di informazioni atte proprio a generare dei dati appositi per la presentazione dei dati ai clienti

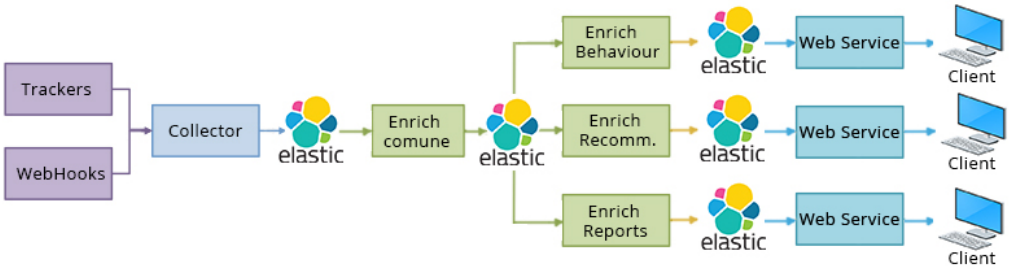

Figura 3.4: Workflow di Snowplow personalizzato da THRON

Tutti questi moduli sono software basati su Apache Spark scritti in scala. Leggono i dati da instanze di Elasticsearch popolate dai moduli precedenti e post elaborazione li salvano in altre istanze di Elasticsearch o in diversi indici dello stesso.
Lavorando a stretto contatto col team di Intelligence ho apportato delle modifiche all'enricher comune, ho aggiunto il supporto ai nuovi tipi di evento che altrimenti erano filtrati. Ho poi aggiunto anche il supporto a nuove informazioni come l'identificativo dell'applicazione necessario per associare agli

eventi da quale applicazione è stato scaturito.
Per il recommendation e il behaviour ho apportato piccolissime modifiche solo per far si che questi software filtrino i nuovi eventi. Questo è infatti necessario al momento perché gli algoritmi di raccomandazione e per l'analisi del comportamento utenti si basano su algoritmi di machine learning delicati e per introdurre i nuovi eventi c'è bisogno di un analisi molto approfondita. Questo era quindi impossibile con i vincoli temporali dello stage e potrebbe essere un progetto di stage futuro.
Nell'enricher dei reports, che è quello che produrrà i dati da interrogare per la creazione dei grafici qualitativi, abbiamo aggiunto il supporto ai nuovi eventi e alle nuove informazioni. Inoltre per rendere più ordinato l'output dei dati abbiamo scelto di scrivere i dati in output su diversi indici di Elasticsearch raggruppati per tipo di evento. Questo può infatti migliorare leggermente le performance delle query se si conosce in partenza il tipo di eventi da cercare. Ci sono stati dei problemi per l'aggiunta di nuove informazioni per la serializzazione di questi ma grazie all'aiuto del team sono riuscito a finire le attività con un piccolo anticipo rispetto ai tempi prefissati.
I web services necessari a interrogare l'Elasticsearch contenente i dati dei reports erano già presenti, è stato necessario effettuare delle modifiche su quest'ultimi per fari si che rendessero disponibili anche i nuovi dati elaborati ma queste modifiche sono state effettuate dal resto del team guadagnando così del tempo.

Seconda Fase

Analisi raccolta del numero di raccomandazioni

Ho partecipato a delle riunioni col team Core per capire come rendere disponibili ai clienti i dati relativi alla quantità di raccomandazioni fatte dalle loro applicazioni. Questo è molto importante poiché i clienti al momento dell'acquisto dello strumento di raccomandazioni possono scegliere diversi Business Model che prevedono una quantità di raccomandazioni oraria massima prefissata. Superata questa soglia lo strumento non raccomanda più secondo algoritmi intelligenti ma propone solo contenuti simili.
THRON possiede un Elasticsearch in cui vengono immagazzinati tutti gli accessi alle loro API. Tuttavia questi dati vengono salvati per un massimo di sessanta giorni poiché la mole di dati è non indifferente (si parla di almeno quattro gigabyte di log di accessi al giorno contando solo gli accessi alle API, senza contare quindi le fruizioni delle cdn_G).
I dati da fornire ai clienti devono avere una retention_G di almeno due anni,

per questo motivo si è deciso di creare un qualche processo che ad intervalli regolari andasse ad interrogare l'Elasticsearch di monitoring, ricavasse tutte le informazioni riguardanti gli accessi alle raccomandazioni e le andasse a salvare in qualche altro database.
I grafici potranno avere una granularità oraria, giornaliera o mensile, ho pensato quindi di salvare di dati raggruppati per queste granularità per avere un risultato immediato quando si effettua una richiesta.
L'Elasticsearch di cui si parla è in una rete privata dell'azienda perché contiene dati che non devono essere resi pubblici, analogamente il database in cui si andranno a salvare i dati sarà nella stessa rete interna è quindi necessario anche creare uno strato di web services che rendano disponibili i dati riguardanti la quantità di raccomandazioni all'esterno.
Come ultima cosa abbiamo pensato anche di creare, nel caso di fosse tempo, un servizio che ripulisse i dati più vecchi di due anni con granularità maggiore di quella mensile. Questo perché è poco auspicabile che un utente voglia vedere le raccomandazioni relative ad un giorno preciso di più di due anni fa.
Inoltre la quantità di dati da salvare non è molto grande ma si parla comunque di una riga su un database per ogni ora per ogni applicazione raccomandativa, facendo dei calcoli si ottengono **8760** entry orarie per applicazione. Non è un numero molto grande ma bisogna pensare che le applicazioni raccomandative potrebbero crescere andando ad aumentare l'ordine di grandezza dei dati da salvare.

Progettazione e sviluppo seconda fase

Per prima cosa mi è stato chiesto di pensare alle firme di tutti i servizi da creare. THRON mantiene tutta l'architettura software in un progetto di Enterprise Architect$_G$ tramite diagrammi delle classi e altro ancora.
Hanno sviluppato un plugin che a partire dai diagrammi crea tutte le interfacce scala automaticamente.
Mi hanno dato l'accesso a diverse loro repositories per farmi familiarizzare con il loro lato server, questo mi ha aiutato molto per la stesura delle firme dei vari metodi, anche perché la serie di web services per l'interrogazione dei reports già esistente era comunque simile a ciò che dovevo fare io. Sfruttando anche l'ereditarietà e i mixin Scala ho potuto riutilizzare parte del loro codice anche per le firme.
Concordate le firme il team Core le ha aggiunte alla loro infrastruttura e ha generato le interfacce. Ho dovuto quindi creare le implementazioni di quest'ultime e per la parte di web services ho seguito molto quelli già esistenti. Questo mi ha facilitato il lavoro, ho potuto riutilizzare componenti di utilità

già scritti da THRON come i servizi per l'autenticazione e dei meccanismi per l'accesso al database.
Si è scelto di usare un MongoDB per il salvataggio dei dati per due motivi:

- Il team Core faceva già un grande uso di questo database, c'erano già quindi classi di utilità per l'utilizzo di quest'ultimo

- I dati da salvare non erano di tipo relazionale quindi era più idoneo usare un database NoSQL

Ho implementato, sempre anche grazie all'aiuto dell'azienda, i servizi per l'estrazione dei dati da Elasticsearch e il salvataggio in MongoDB.
Per fare in modo che il servizio fosse eseguito ogni ora il team Core mi ha fornito l'accesso allo strumento di notifiche amazon, Amazon SNS. Hanno implementato diverse code di sottoscrizione con intervalli orari diversi per ogni necessità. Sottoscrivendo un servizio ad una di queste code Amazon SNS provvede automaticamente a risvegliare, tramite una notifica push$_G$, i servizi sottoscritti.
Ho dovuto quindi tramite configurazione sottoscrivere questo servizio alla coda oraria per far si che fosse schedulato in automatico.
Il scritto il servizio in modo che:

- Per prima cosa legge dal database quando è stata l'ultima elaborazione dei dati

- Interroga Elasticsearch chiedendo i dati relativi alle raccomandazioni dall'ultima elaborazione

- Crea un documento, nella collezione contenente i dati orari, con al suo interno il numero di raccomandazioni fatte per ogni applicazione raccomandativa

- Controlla se è passato più di un giorno dall'ultima elaborazione dei dati giornaliera

- Se si raccoglie, al suo interno, i dati relativi all'ultimo giorno e salva in una collezione adibita a contenere i dati giornalieri un documento con quest'ultimi

- Effettua lo stesso controllo e la stessa logica per i dati mensili

- Aggiorna tutti i timestamp relativi alle ultime elaborazioni fatte

In questo modo nel database sono presenti diverse collezioni contenenti i dati con le granularità richieste, nè il client nè il server devono così effettuare calcoli quando i dati gli sono restituiti.
Ho pensato di organizzare i dati in questo modo anche per facilitare la creazione del servizio di pulizia dei dati vecchi, se fossero tutti i dati in un unica collezione avrebbe dovuto ad ogni esecuzione filtrare quelli con granularità diverse da quella da cercare rendendolo più lento.
Ho poi implementato i web services necessari, è stato possibile riutilizzare tantissimo del codice già presente facilitandomi il compito, ho infatti utilizzato codice ed esteso alcuni dei web services relativi alla fruizione dei dati dei reports finendo così le attività in anticipo rispetto al previsto.
Essendo in anticipo ho scritto il servizio per l'eliminazione dei dati più vecchi di due anni dal database, anche questo non ha creato problemi perché avevamo effettuato la progettazione pensando che la creazione di questo servizio era necessaria prima o poi. Sottoscritto il servizio alla coda mensile di SNS ho implementato facilmente il resto.
Assieme al team Core abbiamo anche implementato un servizio di monitoring che analizzi, ad ogni esecuzione, la qualità del processo eseguito. In particolare nel caso il servizio fallisca lanciando un eccezione il servizio di monitoring avverte il team tramite una notifica su uno schermo che hanno in ufficio che è avvenuto un errore. Manda una notifica anche nel caso il servizio di metta più di una certa durata che abbiamo fissato a venti minuti.
Grazie a questa console il team Core tiene monitorato tutta una serie di altri loro servizio per capire, prima di ricevere eventuali segnalazioni dei clienti, se qualcosa non sta funzionando o se ci sono rallentamenti che necessitano una analisi da parte loro.

Terza fase

Analisi esposizione dati

Lo scopo di questa fase era trovare dei grafici adatti ad esporre in una dashboard i dati raccolti in maniera chiara e concisa.
Sono state fatte delle riunioni con membri del team di Design e di UX sia per capire quali grafici inserire nella dashboard, l'usabilità che essa deve fruire e come integrarla in bacheca$_G$.
Per l'integrazione in bacheca$_G$ mi sono state elencate le tecnologie da adottare mentre mi è stata data l'opportunità di scegliere cosa utilizzare per la creazione dei grafici.
Abbiamo pensato di creare un grafico per la rappresentazione della quantità

di raccomandazioni.
In particolare il grafico presenterà in asse X il tempo, in base alla granularità scelta. Ad esempio potrà essere la lista delle ore, dei giorni o dei mesi.
Nell'asse Y invece sarà presente il numero di raccomandazioni per ogni applicazione slave. Nel caso la granularità mostrata sia oraria sarà presente anche una linea orizzontale che delimita il massimo numero di raccomandazioni effettuabili.
Abbiamo scelto poi di creare più grafici per fornire informazioni sulle singole applicazioni raccomandative.
Un grafico, principale, che fornisca i dati sui cinquanta contenuti più raccomandati. Questo grafico avrà nell'asse Y i contenuti, possibilmente con una Thumbnail per facilitarne il riconoscimento e nell'asse X la quantità di click e di impression.
Due grafici a lato che mostrino le tag dei contenuti più raccomandati e le tag riguardanti i profili associati ai contenuti più raccomandati.
Tutti questi grafici devono essere filtrabili per periodo di tempo, ordinati per impression o per click e in più dev'essere possibile cercare singoli contenuti, questo serve nel caso si sia interessati ad un particolare contenuto che magari non è tra i primi cinquanta in ordine di raccomandazioni.
Al click sulla thumbnail di un contenuto far apparire una finestra modale in cui si possano vedere più dettagli del contenuto scelto, tra cui una descrizione, il tipo di utenti a cui è stato raccomandato e che ci hanno cliccato e gli argomenti trattati dagli utenti che hanno cliccato.

Progettazione e sviluppo terza fase

Per prima cosa ho scelto cosa utilizzare per la creazione dei grafici. I grafici devono essere presentati nella dashboard che consiste in una pagina web che andrà a finire all'interno di bacheca$_G$. Il linguaggio da utilizzare era perciò Javascript, ho studiato e provato tre librerie che mi sembravano le più adatte a creare questi grafici:

- D3js
- FusionCharts
- KendoUI

D3js: (Data Driven Documents) è una libreria gratuita nata nel 2011 per la visualizzazione dinamica e interattiva di dati organizzati, offre funzionalità a basso livello per creare grafici attraverso oggetti SVG.
Si possono avere dei risultati sbalorditivi e di forte impatto, tuttavia io non

dovevo fare grafici particolarmente complessi, in più necessità di molto più codice e tempo. In più la retro compatibilità non è garantita fino a versioni dovevo supportare quindi l'ho esclusa come possibilità.
KendoUI: è un framework a pagamento che offre una grandissima quantità di funzionalità built-in, ma mi sono concentrato nella libreria che offre API per creare grafici. Questa libreria è molto semplice da usare, si raggiungono risultati quasi immediatamente e molto belli. Offre un buon livello di personalizzazione e una sufficiente retro compatibilità.
Tuttavia ho scelto di utilizzare la terza libreria in particolare per questioni di feeling utente. Infatti i grafici dei reports$_G$ THRON utilizzano tutti la libreria FusionCharts.
FusionCharts: è una libreria a pagamento nata nel 2003 e offre una vasta gamma di diagrammi, mappe, widgets e dashboards già fatti con cui arricchire i propri siti o applicazioni web. Dopo averla provata è risultata essere la più facile, quasi non necessità conoscenze di programmazione per avere risultati semplici. Molto belli i grafici prodotti tuttavia meno personalizzabili rispetto a quelli creati con Kendo ad esempio.
La scelta è ricaduta su FusionCharts per mantenere un look identico agli altri grafici di reportistica già fatti da THRON.
La creazione dei grafici è stata molto semplice, l'uso di FusionCharts affiancato a dei web services che rispondono con una struttura dati da me scelta, simile a quella che FusionCharts usa mi ha facilitato il compito. Un altra cosa è stata l'integrazione di questa dashboard in bacheca$_G$.
Essendo che il mio codice andava integrato all'interno del loro software ho dovuto adeguarmi a regole, strumenti e tecnologie utilizzati dal loro team.
Ho utilizzato *Knockout.js* che è un framework per l'implementazione di pattern MVVM$_G$ con templates.
Gulp, simile a Grunt, per automatizzare la compilazione ed altre azioni comuni.
Un altro strumento utile che ho dovuto imparare è stato *tooltipster*. È un plugin per jQuery che facilita la creazione di tooltip$_G$ e la gestione di essi, ad esempio per far si che un tooltip si chiuda quando se ne apre un altro.
La parte che mi ha creato più difficoltà è stata la stilizzazione della dashboard, in bacheca$_G$ utilizzano il framework *SASS* per mantenere il codice CSS più pulito. Questo aggiunge funzionalità come variabili ed ereditarietà ai css e il team utilizza un pattern MVC$_G$ in tutti i loro CSS. È stato difficile entrare in sintonia con le loro convenzioni poiché non avevo mai avuto modo di rapportarmi con progetti così complessi. Anche a fine stage non mi reputo ancora in grado di destreggiarmi facilmente in quel progetto ma grazie anche al supporto che mi è stato dato sono riuscito a mantenere una struttura analoga alla loro.

Credevo di avere più problemi per il grande supporto a browser richiesto ma grazie all'uso di molti framework e librerie non ho avuto problemi di questo tipo.
Di seguito troviamo delle immagini raffiguranti l'output che un cliente si ritrova. Le immagini sono state prese da degli ambienti di test THRON.
In Figura 3.4 possiamo vedere il grafico che mostra, per una scelta applicazione master raccomandativa, la quantità delle raccomandazioni con granularità oraria e il limite massimo previsto dal proprio business model.

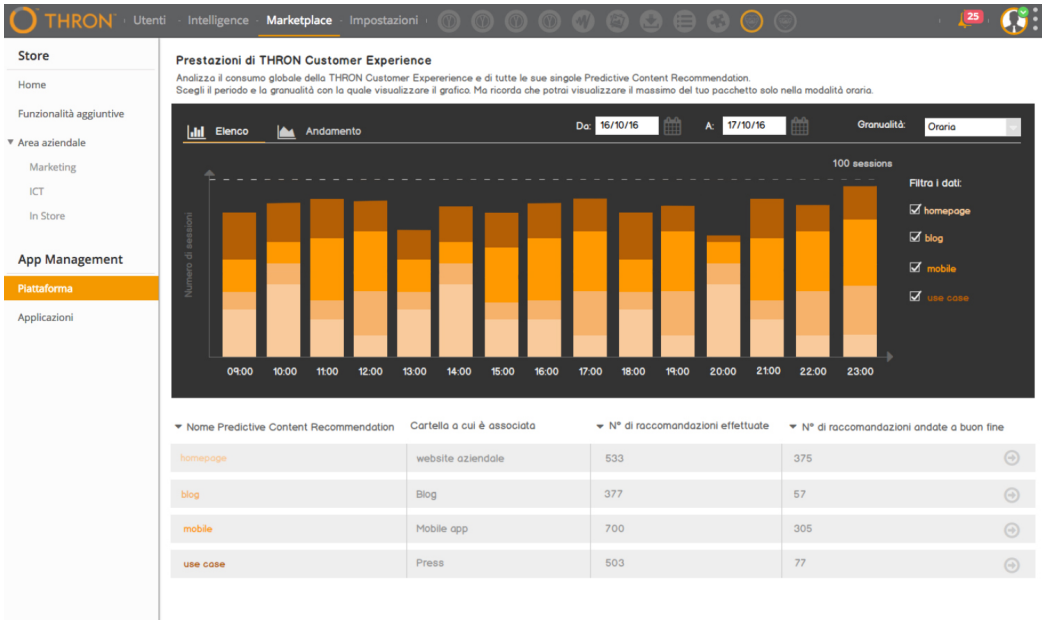

Figura 3.5: Grafico quantitativo visibile all'interno della dashboard di una applicazione raccomandativa master

In Figura 3.5 possiamo vedere il grafico che mostra, per una scelta applicazione raccomandativa slave, una diagramma a barre con i valori delle impression e click per vari contenuti. Alla destra abbiamo un riassunto in cui sono mostrate le impression totali dell'applicazione, i click totali e due grafici a tag cloud in cui sono presenti nel primo le tag relativi la classe target dei contenuti, mentre in quello sottostante le tag relativi la classe topic dei contenuti.

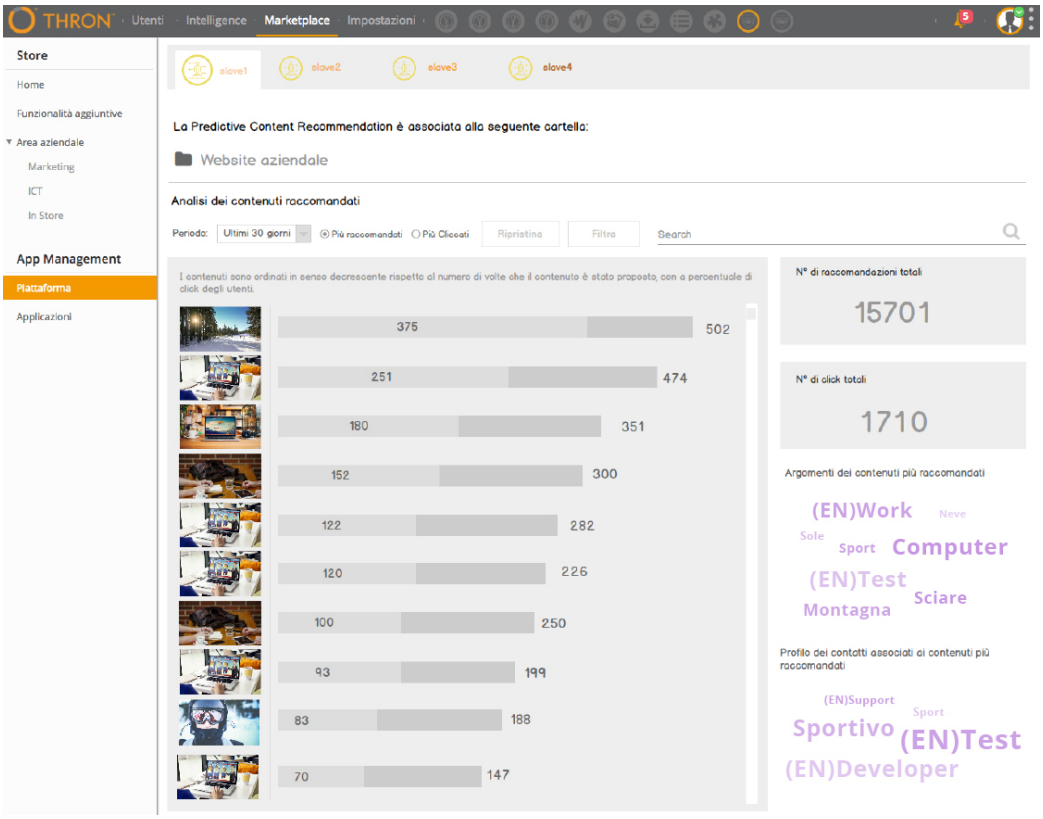

Figura 3.6: Grafico qualitativo visibile all'interno della dashboard di una applicazione raccomandativa slave

Al click sulla thumbnail$_G$ di un contenuto viene aperta una pagina modale che possiamo vedere in Figura 3.6.

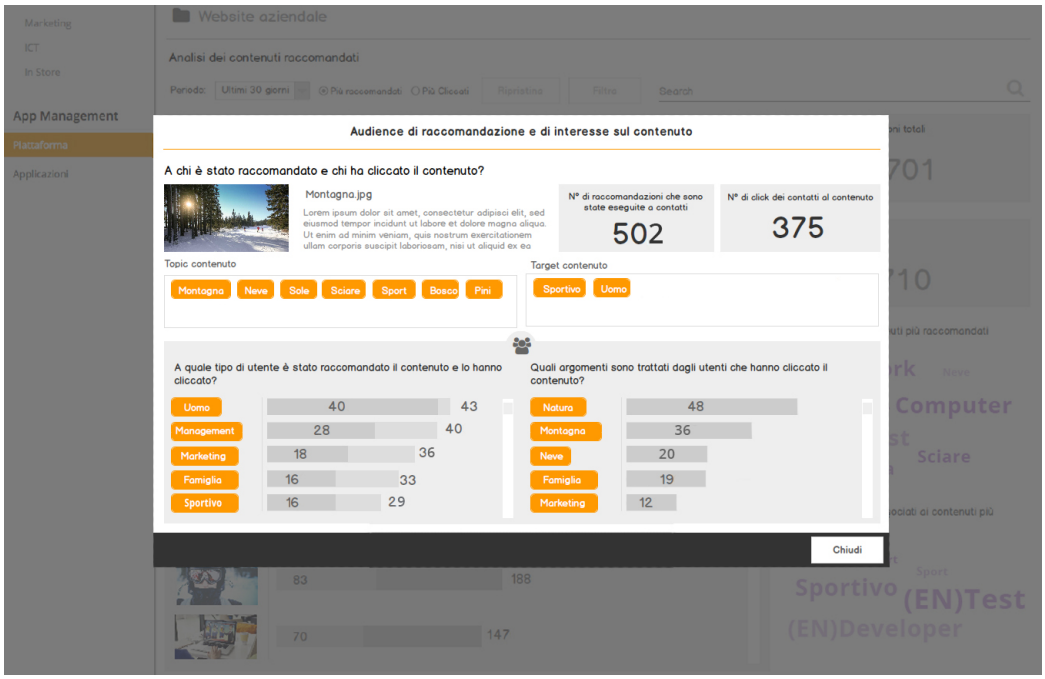

Figura 3.7: Finestra modale aperta al click sul primo contenuto del grafico visto in Figura 3.5

Nella finestra che viene presentata all'utente si può vedere un riassunto sull'audience dell'utenza a cui è stato proposto il contenuto e che ha espresso interesse.
In particolare oltre ad una descrizione del contenuto e al numero di impression e click ricevuti troviamo informazioni come le tag del contenuto. Sia le tag della classe topic che parlano del contenuto sia quelle della classe target che indicano verso quale tipologia di utenti è orientato il contenuto.
In basso infine è presente un grafico a barre per mostrare un elenco delle tag relative agli utenti a cui è stato proposto il contenuto. Questo è interessante per capire, per le varie tipologie di utenti a cui è stato proposto, quali categorie sono state più interessate.
In basso a destra, invece, un altro grafico a barre che mostra quali tipi di contenuto sono di interesse alle tipologie di utenti a cui è stato proposto il contenuto.
Tutte queste informazioni, oltre a poter dare un indicazione sulla qualità e quantità delle raccomandazioni, può dare spunti per strategie di marketing o di business. Ad esempio può essere interessante vedere che un contenuto che ha come target una certa tipologia di utenti poi susciti interesse non in quella categoria ma bensì in un altra. Potrebbe essere un segnale che indichi che la tipologia assegnata non sia quella giusta o che il contenuto sia fuori

tema.
Alla fine di queste attività era finito l'intero periodi di stage per cui non è stato possibile realizzare il meccanismo per la personalizzazione dei widgets di raccomandazione.

Obiettivi raggiunti

Prima fase - Obbligatori

OBIETTIVO	STATO
Dovrà essere aggiunta la funzionalità di tracciare impression$_G$ nella libreria dei tracciamenti THRON	Soddisfatto
Dovrà essere aggiunta la funzionalità di tracciare click$_G$ nella libreria dei tracciamenti THRON	Soddisfatto
Dovrà essere aggiunto un meccanismo per tracciare eventi come click$_G$, load$_G$ e glossimpression in automatico nella libreria dei tracciamenti THRON	Soddisfatto
Le modifiche da apportare alla libreria dei tracciamenti dovranno mantenere una retro compatibilità totale	Soddisfatto
Le modifiche devono essere supportato in Internet Explorer 9 e successivi	Soddisfatto
Le modifiche devono essere supportato in Firefox 3.5 e successivi	Soddisfatto
Le modifiche devono essere supportato in Chrome tutte le versioni	Soddisfatto
Le modifiche devono essere supportato in Safari OSX5 e successivi	Soddisfatto
Le modifiche devono essere supportato in Android 4.1 e successivi: Chrome tutte le versioni	Soddisfatto
Le modifiche devono essere supportato in IOS 4.x e successivi: Safari 4 e successivi, Chrome tutte le versioni	Soddisfatto
Dovrà essere modificato l'enricher per aggiungere il supporto ai nuovi eventi	Soddisfatto
Aggiungere al modulo Reports il supporto ai nuovi eventi	Soddisfatto
L'enricher Reports dovrà salvare tutti gli eventi in indici diversi raggruppati per tipo di evento	Soddisfatto
Far si che gli altri moduli di Intelligence ignorino questi nuovi eventi	Soddisfatto

Prima fase - Desiderabili

OBIETTIVO	STATO
Il meccanismo contenente la logica di automazione degli eventi dovrà essere aperto ad estensioni per futuri eventi	Soddisfatto

Seconda fase - Obbligatori

OBIETTIVO	STATO
Creare un servizio che interroghi Elasticsearch di monitoring, ricavi i dati degli accessi alle raccomandazioni con granularità oraria e li salvi in un database	Soddisfatto
Creare un servizio che interroghi Elasticsearch di monitoring, ricavi i dati degli accessi alle raccomandazioni con granularità giornaliera e li salvi in un database	Soddisfatto
Creare un servizio che interroghi Elasticsearch di monitoring, ricavi i dati degli accessi alle raccomandazioni con granularità mensile e li salvi in un database	Soddisfatto
Progettare e sviluppare dei web services interrogabili dall'esterno che permettano di interrogare il database	Soddisfatto
Utilizzare delle politiche di accesso e un sistema di permessi uguali agli altri web services THRON per questi nuovi	Soddisfatto

Seconda fase - Desiderabili

OBIETTIVO	STATO
Sviluppare un servizio che ripulisca i dati, con granularità oraria e giornaliera, più vecchi di due anni	Soddisfatto

Terza fase - Obbligatori

OBIETTIVO	STATO
Realizzare un grafico che indichi il numero di raccomandazioni fatte da un applicazione raccomandativa master	Soddisfatto
Dare la possibilità di scegliere la granularità temporale per il grafico, in particolare granularità oraria, giornaliera e mensile	Soddisfatto
Dare la possibilità di filtrare i risultati del grafico scelta una data di inizio e di fine	Soddisfatto
Dare la possibilità di filtrare i risultati del grafico scelta una o più applicazioni raccomandative slave	Soddisfatto
Creare un grafico per ogni applicazione slave che mostri i cinquanta contenuti raccomandati visti più volte, mostrando anche la quantità di click ricevuti	Soddisfatto
Creare un grafico che mostri gli argomenti relativi ai contenuti più visti	Soddisfatto
Creare un grafico che mostri il profilo dei contatti associato ai contenuti più visti	Soddisfatto
Dare la possibilità di visualizzare una descrizione di un contenuto al click su di esso	Soddisfatto
Dare la possibilità di visualizzare le tag di un contenuto al click su di esso	Soddisfatto
Dare la possibilità di visualizzare le tipologie degli utenti che hanno interagito con un al click su di esso	Soddisfatto
Dare la possibilità di visualizzare gli argomenti più trattati da queste tipologie di utenti al click su di un contenuto	Soddisfatto
Dare la possibilità di filtrare per un periodo qualunque tutti i grafici relativi ad una applicazione slave	Soddisfatto
Dare la possibilità di cercare un contenuto specifico in tutti i grafici relativi ad una applicazione slave	Soddisfatto
Dare la possibilità di ordinare i contenuti per più visti o più cliccati sul grafico che mostra i cinquanta contenuti di una applicazione slave	Soddisfatto

Terza fase - Desiderabili

OBIETTIVO	STATO
Sviluppare una sezione in cui vengono dati consigli sulle raccomandazioni, ad esempio consigliare quali contenuti si consiglia di togliere dalle possibili raccomandazioni	Non soddisfatto
Sviluppare un meccanismo per la personalizzazione dei widget di raccomandazione	Non soddisfatto

Valutazione retrospettiva

Obiettivi raggiunti

L'azienda possedeva chiari obiettivi che volevano fossero raggiunti, finito lo stage è stato possibile determinare il grado di completezza raggiunto col progetto.
Gli obiettivi riguardanti la prima parte dello stage sono stati tutti raggiunti, obbligatori e non. Nonostante queste attività abbiano provocato dei ritardi rispetto al piano di lavoro, rendendo così impossibile finire gli ultimi obiettivi facoltativi, l'azienda è stata molto soddisfatta del lavoro fatto. In particolar modo del modulo aggiuntivo per il tracciamento automatico delle immagini. Questo infatti da l'inizio ad un progetto molto importante poiché l'azienda ha come obiettivo, non a lungo termine, di produrre una libreria in grado di tracciare automaticamente tutti gli eventi, relativi a contenuti THRON in pagina, senza alcuno sforzo necessario da parte del cliente.
Per quanto riguarda la seconda fase, anche in questo caso l'azienda si è dimostrata soddisfatta degli obiettivi raggiunti. Apprezzato il fatto che si sia creato anche un servizio per ripulire i dati, questo infatti era particolarmente richiesto dal team Core essendo il database in cui scrivevo i dati sotto la loro responsabilità.
La terza fase, a causa dei ritardi provocati dalle attività dei tracciamenti, è stata la meno soddisfacente. Gli obiettivi obbligatori sono stati tutti raggiunti con soddisfazione, tuttavia gli opzionali non sono stati sviluppati.
Durante le attività di analisi relative a quest'ultima parte si è però pensato ed aggiunto uno degli obiettivi facoltativi ovvero sviluppare una sezione in cui vengono dati dei consigli sulle raccomandazioni. Questo obiettivo può in futuro, in seguito a ulteriori analisi, diventare un progetto di stage a se stante molto interessante. Infatti si è pensato che sarà possibile creare una box, al di sotto della dashboard, che dia consigli intelligenti sulle raccomandazioni, andando a suggerire quali contenuti è meglio togliere e quali invece si potrebbero aggiungere presentando ulteriori statistiche più avanzate.
Per quanto riguarda gli obiettivi personali, mi ritengo pienamente soddisfatto

dell'esperienza fatta. L'uso di diverse tecnologie che mi ha spinto a scegliere questo progetto è stato più che soddisfacente, inoltre ho affrontato problemi che non avrei mai pensato di affrontare nè di risolvere, grazie anche all'aiuto dell'azienda.
Il progetto comunque è stato considerato molto positivo e la maggior parte delle modifiche era in produzione durante la stesura della mia tesi, dandomi così un ulteriore motivo di orgoglio.

Competenze acquisite

Inizio ringraziando l'università che grazie al corso mi ha fatto acquisire competenze senza le quali non sarei mai riuscito ad affrontare un progetto di questa portata. Le competenze tecniche fornitemi dall'università sono state più che sufficienti per affrontare i problemi di programmazione che mi sono stati sottoposti.
Inoltre grazie ai corsi orientati al lavoro di gruppo, come Ingegneria del Software, mi è stato facilitato il compito di lavorare a stretto contatto con persone diverse e ad adeguarmi alle norme dei vari team con cui ho collaborato.
Riguardo alle competenze tecniche acquisite, ho sviluppato molto la conoscenza su linguaggi di programmazione che già conoscevo come Scala e Javascript, utilizzando moltissime librerie e andando a lavorare in progetti non del calibro di quelli universitari.
Le attività che mi hanno colpito e formato maggiormente sono state quelle relative la prima fase. Non avevo nessuna competenza riguardante l'attività di profilazione e di tracciamento degli utenti e devo dire che mi ha sorpreso tantissimo capire come questi sono fatti e quali sono i problemi che sono stati affrontati dall'azienda per rendere possibile un sistema di tracciamenti molto complesso che funzioni in ogni browser. Particolarmente difficile infatti l'implementazione per i browser vecchi come Internet Explorer 9 e quelli non permissivi come Safari.
Ho compreso anche molti aspetti del linguaggio Javascript come il concetto di ereditarietà prototipata e l'uso di tantissime librerie a me prima sconosciute.
Per quanto riguarda la parte server sono stato contento di riutilizzare il linguaggio Scala che trovo sia un linguaggio sempre più in crescita grazie appunto alla possibilità di scalare orizzontalmente. In particolar modo mi ha colpito l'utilizzo dei servizi Amazon che non avevo mai compreso appieno non lavorando in progetti che necessitano l'elaborazione di grandi quantità di dati.
Sono stato molto sorpreso anche dalle attività dell'ultima fase, che pensavo

molto più semplici. La manutenzione di un progetto così vasto lato user experience infatti necessità l'utilizzo di framework che non conoscevo e delle rigide regole per evitare che si crei poi molto codice impossibile da riutilizzare o da modificare.
Per ultima cosa, ma sicuramente la più importante, ho apprezzato moltissimo interfacciarmi col mondo del lavoro e capire come effettivamente funzioni un azienda e come si lavori in un team aziendale con molta esperienza. Mi ritengo fortunato che grazie a questo progetto ho dovuto interagire con quattro team differenti andando ad avere quindi nozioni diverse da diverse persone.

Glossario

Ascendant In linguaggio HTML gli ascendant di un elemento sono tutti gli elementi che stanno al di sopra dell'interessato come gerarchia. In una pagina HTML quindi tutti gli elementi in pagina hanno come ultimo ascendant l'elemento HTML.

Bacheca È il software interfaccia web per l'uso della piattaforma THRON. Scritto interamente in Javascript, HTML e CSS espone tutte le funzionalità offerte da THRON tramite una semplice applicazione web.

CDN Acronimo di Content Delivery Network. Consiste in una rete distribuita di server in diversi data center. Serve per offrire all'utente finale alte performance nell'erogazione di risorse.

Click Quando si parla di click in questo elaborato si intende l'evento click da tracciare. Questo, come si può intuire, è associato al click sull'elemento HTML stesso.

CMS Acronimo di Content Management System. Sono applicazioni software per la creazione, gestione e modifica di contenuti digitali attraverso semplici interfacce. Uno dei CSM più famosi è WordPress.

Codice servizio In THRON il codice servizio è un'istanza della piattaforma configurata per uno specifico utente, ad ogni istanza è associata una stringa per identificarla.

Data scientist Una figura, relativamente recente, che ha come compito quello di analizzare e interpretare dati. Nel caso di THRON è una figura che, tramite i dati di intelligence, deve capire come sviluppare algoritmi per migliorare certi aspetti di intelligence, ad esempio il motore raccomandativo.

Div In HTML è un tag che viene usato per dividere sezioni della pagina o per raggruppare elementi che hanno caratteristiche simili.

DOM Acronimo di Document Object Model. È uno standard che definisce l'accesso e la modifica di contenuti, strutture e stile di un documento. Il DOM HTML è uno standard che definisce gli elementi HTML come oggetti, le proprietà, i metodi di accesso e gli eventi di questi.

Enterprise Architect Software per la modellazione di diagrammi UML

Impression In questo elaborato, concordato con THRON, per impression si intende l'evento che avviene quando un particolare elemento HTML entra nella viewport$_G$ di un utente durante la navigazione.

Load In questo elaborato, concordato con THRON, per load si intende l'evento che avviene quando un particolare elemento HTML viene caricato nella pagina di un utente durante la navigazione.

Lucene Apache Lucene è un software scritto in Java per la ricerca di testo ad alte prestazioni.

Metadati In THRON i metadati sono delle informazioni aggiuntive che si possono associare a contenuti ed utenti. Simili alle tag servono per arricchire con informazioni.

Mutation observer Danno agli sviluppatori un'opportunità di reagire, tramite callback, a cambiamenti del DOM$_G$

MVC Acronimo di Model View Controller. È un design pattern software architetturale per dividere logicamente un applicativo in tre macro parti: il Model che gestisce i dati, la logica e le regole dell'applicativo, la View che consiste nella rappresentazione in output delle informazioni e il Controller che accetta input e li converte in comanda per gli altri due moduli.

MVVM Acronimo di Model View View-Model. È un design pattern software architetturale. Divide il software in tre macro parti: il Model che contiene il modello dei dati, la logica per accedere e per modificarli, la View che consiste nella rappresentazione in output delle informazioni e il View-Model. Quest'ultimo è un astrazione della View che espone metodi e comandi. In questo pattern per ogni View va effettuato un binding con un View-Model che ne descrive le operazioni possibili.

Notifiche push Le notifiche push, al contrario delle notifiche pull nella quale un client deve richiedere informazioni ad un server, sono informazioni inviate da un server. Tipicamente il client deve essere sottoscritto in qualche modo per riceverle.

Parser Per parser si intende un software che effettua un attività di parsing, ovvero effettua analisi di dati secondo regole formali. Nel caso specifico il parser effettua l'analisi di elementi HTML per cercare elementi da tracciare in pagina.

Pipeline In ambito software consiste in una catena di processi per produrre un output desiderato. In questo elaborato per pipeline si intende la serie di processi che portano dal tracciamento degli eventi ai dati finali da esporre ai clienti.

Polyfill È del codice che implementa feature su web browser che non supporta quest'ultima. Serve quindi a dare una retro compatibilità, per specifiche funzionalità, a browser non più mantenuti.

Project Management disciplina che consiste nell'iniziare, pianificare e controllare il lavoro di un team per raggiungere specifici obiettivi prefissati.

Promise Sono oggetti usati per la programmazione asincrona. Rappresenta un valore che può non essere ancora disponibile, ma che può esserlo in futuro.

Retention Definisce le politiche di persistenza dei dati per richieste legali e business. In questo elaborato si intende la quantità, obbligatoria per contratto, di tempo per cui i dati devono essere resi disponibili al cliente.

Reports In THRON i Reports rappresentano i dati, elaborati da una pipeline, che contengono tutte le informazioni necessarie per essere esposti al cliente, di solito interpretati in un grafico.

Seek In questo elaborato per Seek si intende l'evento che avviene quando un video, tramite un player THRON, viene scorso in avanti o indietro.

Sistema di ticketing un sistema software per la gestione e la manutenzione di issues, utile per mantenere ordine e una gerarchia nei problemi riscontrati nel proprio software o nei propri processi.

Tag In THRON le Tag sono delle entità da associare a contenuti o utenti per classificarli. Le tag sono organizzate con strutture ad albero perciò un contenuto associato ad una tag 'figlio' eredita automaticamente tutta la gerarchia del tag 'padre'.
Inoltre ci sono diverse classi di tag. Per esempio la classe TOPIC di un contenuto contiene tutte le tag che descrivono l'argomento del

contenuto mentre la classe TARGET contiene le tag che descrivo a chi è indirizzato questo contenuto.

Thumbnail In THRON, una thumbnail è un immagine creata a partire da ogni contenuto.

Tooltip È una piccola interfaccia grafica che contiene informazioni o suggerimenti riguardanti un argomento. Di solito le tooltip vengono mostrate quando un utente esprime interesse su un contenuto web cliccando o passando il mouse sopra un elemento.

Viewport È un area virtuale usata dai motori di rendering dei browser per determinare come scalare e ridimensionare contenuti in pagina. Rappresenta l'area che l'utente, durante la navigazione web, vede a schermo.

Widget Per widget di solito si intende un applicativo software semplice e pronto all'uso o un componente software ready to use da aggiungere al proprio applicativo.
I widgets di cui si parla in questo elaborato sono dei piccoli software javascript da copia incollare nelle proprie pagine web per aggiungere funzionalità.

Bibliografia

|1| Snowplow wiki. URL https://github.com/snowplow/snowplow/wiki

|2| Cay Horstmann. Scala for impatient. Url http://horstmann.com/scala

|3| Documentazione ufficiale Apache Spark. URL http://spark.apache.org/docs/1.5.2

|4| Documentazione ufficiale Elasticsearch. URL https://www.elastic.co/guide/index.html

|5|