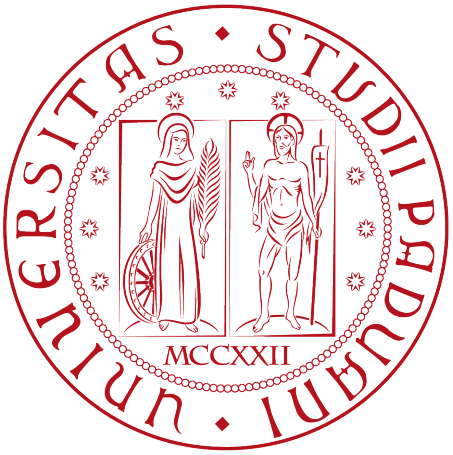
Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



Servizio di ricerca full-text con Elasticsearch e MongoDB

Tesi di laurea triennale

Relatore

Prof. Tullio Vardanega

Laureando

Zanon Edoardo

ANNO ACCADEMICO 2016-2017

“Controlling complexity is the essence of computer programming.”

— Brian Kernighan

Ringraziamenti

Innanzitutto, vorrei esprimere la mia gratitudine al Prof. Varganega, relatore della mia tesi, per l’aiuto e il sostegno fornitomi durante la stesura del lavoro.

Desidero ringraziare con affetto i miei genitori per il sostegno, il grande aiuto e per essermi stati vicini in ogni momento durante gli anni di studio e per aver finanziato la mia carriera universitaria.

Ho desiderio di ringraziare poi i miei colleghi universitari per tutti i bellissimi anni passati insieme e le mille avventure vissute. Li ringrazio inoltre per il sostesto e le sessioni di studio cooperativo, senza del quale avrei proceduto molto più lentamente.

Padova, Set 2017 Edoardo Zanon

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Contesto aziendale

1.1 Profilo aziendale

THRON è un'azienda nata dalla New Vision, una digital media company impegnata nella ricerca e nello sviluppo di tecnologie per la comunicazione in tempo reale. Essa era stata fondata nel 2000 da Nicola Meneghello e Dario De Agostini, con lo scopo di convincere le aziende ad usare Internet come principale mezzo di comunicazione. L'azienda sin da quando è nata ha sempre avuto un grande interesse per le applicazioni web, che non hanno bisogno di installazioni, ma sono accessibili direttamente da browser e da qualsiasi dispositivo diventando così multi-piattaforma.
In partenza ha collaborato con Macromedia e Adobe per sviluppare uno strumento in flash per le videoconferenze multiple che però non è più mantenuto.
Nel 2010 New Vision ha lanciato **4ME**, una piattaforma 'cloud based', che consente il controllo, la condivisione e la pubblicazione di qualsiasi contenuto multimediale, in tutti i canali digitali mediante un'unico punto di ditribuzione. Questo progetto, qualche anno dopo è diventato il nucleo del prodotto attuale, THRON. L'azienda ha cambiato nome, ed è stata ristrutturata dopo un recente aumento di capitale. A partecipare a quest'operazione sono stati finanziatori di grosso calibro, che hanno creduto nel progetto THRON.
THRON ora è cresciuta, e guarda al mercato internazionale, con più di 4 sedi, anche estere. Attualmente nell'azienda lavorano più di 40 persone e l'età media aziendale è di soli 32 anni. Quest'ultimo dato indica la preferenza di THRON nel cercare giovani appena formati. In quest'ambito l'azienda ha già partecipato più volte all'iniziativa STAGE-IT, vincendo anche il premio come miglior progetto per ben 3 volte.

1.1.1 Prodotti

Il prodotto sul quale l'azienda verte è il DAM THRON. Un DAM, acronimo di Digital Asset Management, è un archivio centralizzato che le aziende possono sfruttare per archiviare e gestire i propri contenuti, cioè immagini, documenti, audio, video e qualsiasi tipo di file. Questo prevede le classiche funzioni per la gestione dei contenuti come ricerche,

modifiche alle proprietà, ma anche funzionalità più avanzate.
Quello che differenzia il prodotto di THRON dai DAM di molti competitors, è la sua capacità di condividere gli asset digitali che ospita, su qualsiasi canale evitando duplicazioni, cioè l'utilizzo dello stesso contenuto sui diversi canali presidiati dall'azienda.

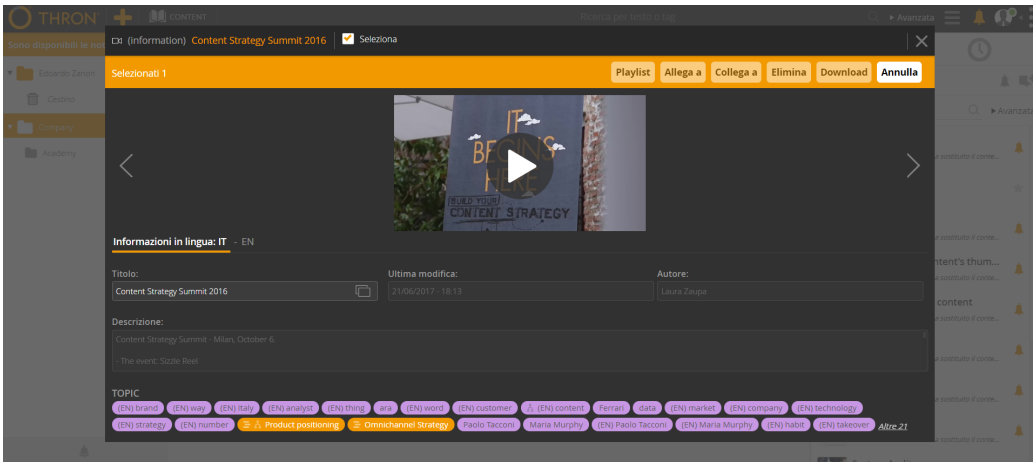


Figura 1.1: Modifica contenuto - THRON

Questo aspetto, oltre a permettere efficienza in termini operativi durante le modifiche, permette di raccogliere informazioni più coerenti riguardo all'uso dei contenuti da parte degli utenti. Queste informazioni vengono poi rielaborate in report utilizzabili dal cliente per scopi commerciali e/o istituzionali.
Durante il caricamento, dopo una prima parte di analisi del contenuto, THRON arricchisce le informazioni con tag e metadati. Questo avviene con un potente motore semantico che aggiunge le informazioni al contenuto subito dopo il caricamento e durante tutto il ciclo di vita, aggiornandole nel tempo. Inoltre è possibile arricchire manualmente il contenuto, aggiungendo, modificando o eliminando le informazioni già presenti. In questo modo i contenuti sono ordinati secondo tag ed è più semplice organizzarli e gestirli, soprattutto quando la mole di dati è elevata. I metadati invece serviranno ai motori di ricerca e ad altre funzionalita per la classificazione di questi.

Ad ogni contenuto è possibile aggiungere dei permessi (ACL), e decidere quindi:

- Gli utenti persone o i gruppi di utenti che possono visualizzare quel contenuto;
- Gli utenti persone o i gruppi di utenti che possono modificare quel contenuto;
- Gli utenti persone o i gruppi di utenti che possono amministrare quel contenuto;

Molte delle azioni che vengono effettuate sulla piattaforma, come modifica di utenti, permessi o assets, vengono tracciate, memorizzando l'azione eseguita per motivi di versionamento.
Il prodotto è fornito di connettori per vari siti e servizi esterni, che consentono l'interfacciamento senza modifiche alla piattaforma. Per esempio all'interno di THRON, è possibile installare connettori per i social network come Facebook e Twitter o di CMS come

WordPress. E' proprio grazie all'utilizzo di questi strumenti aggiuntivi, che è possibile sfruttare la distribuzione Omnichannel del contenuto.

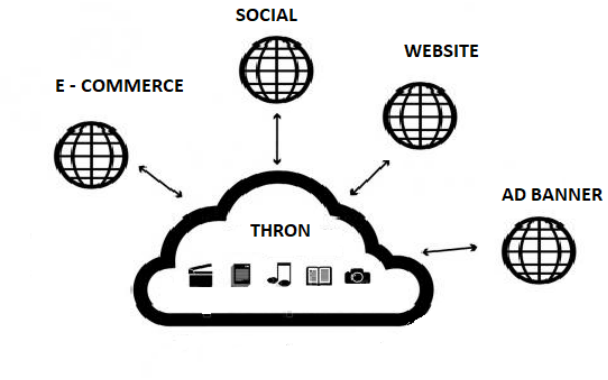


Figura 1.2: Distribuzione informazioni - THRON

Partendo da questo software, l'azienda ha potuto sviluppare il proprio buisness,basandolo non solo sulla vendida esclusiva dello stesso, ma anche su personalizzazioni e sulla fornitura di servizi riguardanti lo streaming e l'elaborazione degli asset digitali per i vari clienti. Il prodotto è affiancato da una serie di applicazioni e servizi fruibili mediante l'apposito marketplace.

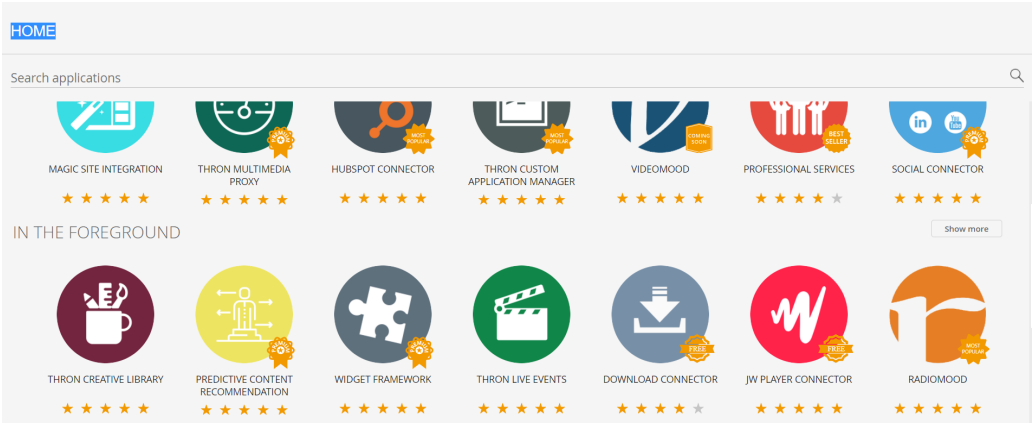


Figura 1.3: Marketplace THRON - (https://marketplace.thron.com/(2017/05/08))

In particolari situazioni, dove i servizi disponibili non sono sufficiente a soddisfare le esigenze del cliente, THRON si propone di creare una soluzione su misura, mediante lo sviluppo di apposite applicazioni.

1.1.2 Propensione all'innovazione

L'azienda è sempre stata all'avanguardia sulle tecnologie di sviluppo, e la sua propensione all'innovazione è stata ulteriormente stimolata dalla migrazione al cloud amazon di maggiorparte dei servizi. Dopo questo passo si è capita la convenienza di un'architettura cloud based, e proprio AWS che propone strumenti sempre innovativi ha aiutato

la crescita di THRON in quest'ambito. L'acquisizione di nuovi talenti e di personale con la passione di sperimentare sempre nuove soluzioni, ha fatto nascere un progetto di aggiornamento continuo, mediante seminari periodici e momenti di confronto sulle soluzioni già adottate o da addotarsi tra colleghi.
Un'ulteriore spinta all'innovazione è data dal mercato nel quale THRON posiziona i suoi prodotti. Essendo quello dei DAM un settore in continua evoluzione, nel cui ambito operano competitor di grande rilevanza, il prodotto per rimanere valido deve a sua volta evolvere e offrire nuove soluzioni alle esigenze dei clienti.
Tutti i suggerimenti, le critiche dei clienti, la continua crisceta del bacino di utenza e le complessità che ne derivano, spingono l'azienda a continuare nel miglioramento del suo prodotto e quindi anche delle proprie conoscenze. Testimonianza di questo, è l'attività profusa in questi ultimi anni, per trasformare il monolite 4me in un prodotto orientato ai microservizi. Per espressa voltà dei vertici aziendali, nell'ambito di questo processo di miglioramento continuo, non solo al software ma anche della struttura aziendale,sono sempre stati mantenuti inalterati l'identità aziendale e le caratteristiche peculiari del prodotto.

1.2 Organizzazione aziendale

1.2.1 Suddivisione organico

Essendo l'azienda molto articolata e suddivisa in diverse sedi, anche estere, non ho potuto approfondire la conoscienza dell'intera organizzazione, bensì solo la parte Executive, ed in particolare il reparto tecnico.
La parte Executive dell'azienda si suddivide in 4 reparti:

- Reparto tecnico;
- Reparto marketing;
- Reparto commerciale;
- Reparto amministrativo.

Il reparto tecnico, ovvero l'area alla quale ero assegnato, è suddiviso in due team. Ad ognuno di questi, sono affidate parte delle funzionalità del prodotto da sviluppare e manutentare. I due team sono rispettivamente:

- Team DAM: copre tutto il prodotto DAM, gestione/contribuzione contenuti, utenti, gestione tag, billing, fruizione attraverso il player e produzione delle applicazione per il marketplace;
- Team CS: copre tutta la parte di registrazione delle visite, contatti anonimi e profilati, reportistica di analisi e le analisi semantiche dei contenuti.

Ogni team è organizzato con 3 figure professionali, con compiti distinti. Queste collaborano senza particolari formalità, ma tracciando sempre le decisioni prese, attraverso documentazione o issue/task. Le figure professionali sono quindi:

- Team Leader: segue le analisi funzionali per gli sviluppi futuri;
- Operation Manager: gestisce le attività di sviluppo del team, ed è la figura di riferimento in caso di problematiche da parte degli sviluppatori;
- Developers: a seconda della propria specialità possono essere più orientati allo sviluppo Server Side, Client Side o Infrastructure.

Esistono anche figure di supporto chiamate "Competence Leader"; Quest'ultime sono trasversali ai team, e servono da riferimento per scelte o difficoltà su un particolare ambito. I principali sono :

- UX competence leader: riferimento per la user expirience;
- Core competence leader: riferimento per la progettazione e le competenze trasversali.
- Ops competence leader: riferimento per le competenze di sviluppo o tecnologiche.

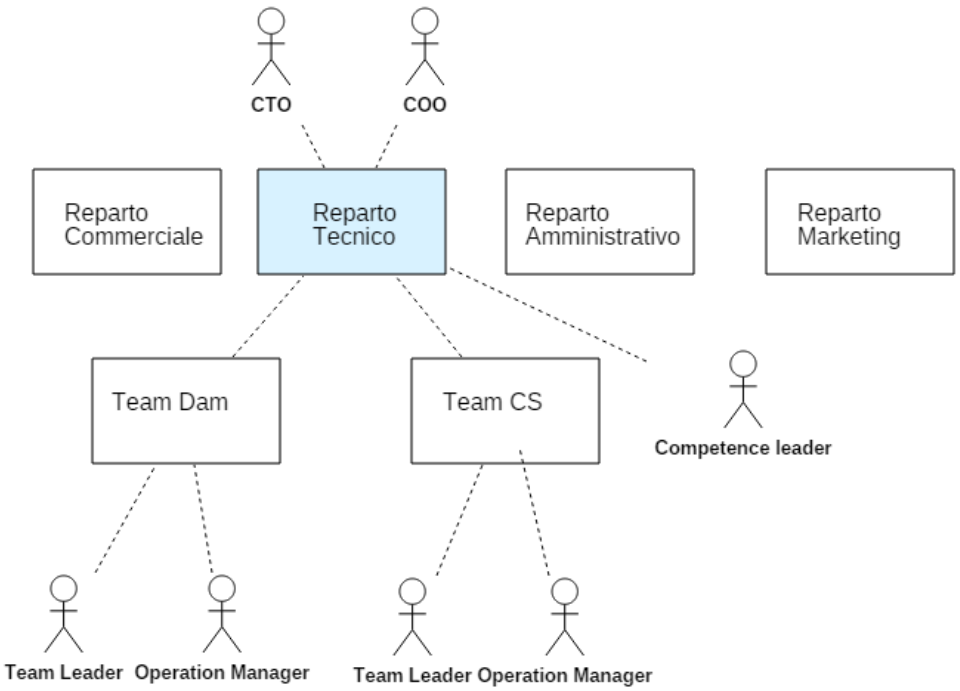


Figura 1.4: Struttura aziendale - THRON

1.2.2 Metodologie

1.2.2.1 Metodi di sviluppo

Modello di sviluppo THRON per lo sviluppo del suo software non utilizza un modello di sviluppo standard. Il modello di sviluppo applicato può essere paragonato ad un modello agile di tipo Scrum. Il software ormai ha una base consolidata da anni, chiamata core, e lo sviluppo avviene mediante passi incrementali, che ne aggiungono funzionalità, o ne migliorano le presistenti, con la riprogettazione.
Periodicamente vengono effettuate delle riunioni con la direzione marketing. Quest'ultima, più a contatto con il cliente, proporrà le modifiche o le funzionalità richieste dai clienti. L'obiettivo della riunione è di capire quali funzionalità è conveniente sviluppare, nell'interesse del *business model*.
Una volta decise le funzionalità da sviluppare, queste vengono assegnate ad un team (del reparto tecnico).
Segue un processo di verifica della fattibilità, nella quale vengono elaborati stime e spesso un Proof of concept, che vengono analizzate in apposite riunioni. Una volta verificato che la funzionalità è sviluppabile ed integrabile nella piattaforma, viene assegnato un task al team di sviluppo più opportuno.

Distibuzione del codice Esistono principalmente due modi per apportare una modifica al codice:

- Sviluppo;
- Hotfix.

Al completamento dello sviluppo, il codice deve essere fornito anche dell'adeguata documentazione. Durante il normale sviluppo del software il codice, viene prodotto in un'ambiente chiamato "sviluppo". Successivamente, una volta testato e funzionante, viene effettuata una richiesta di test nell'ambiente "quality". L'operation manager effettua l'inserimento una volta pronto il pacchetto di funzionalità previsto. Il periodo di test nell'ambiente "quality" dura circa due settimane. In seguito la modifica verrà implementata nell'ambiente "produzione", risulta immediantamente quello fruibile all'utente finale, immediatamente in quanto il prodotto è una web application.
La procedura Hotfix invece viene utilizzata quando la modifica al codice è necessaria per correggere un'errore molto influente e permette di saltare il periodo di quality, e quindi di testing approfonditi. Questo tipo di procedura, deve essere richiesta dal project manager o dall'operation manager in soli casi di urgenza.

Figura 1.5: Processo di modifica

1.2.2.2 Coordinamento

Il coordinameto del lavoro avviene mediante l'utilizzo di task memorizzati sullo strumento YouTrack.
Esistono diverse tipologie di Issue ed ogni tipologia ha una procedura standard per l'attuazione, per il periodo di quality e per la procedura di deploy. I membri dei vari team inoltre, sono liberi di effettuare riunioni e allineamenti indipendenti; Le principali però sono convocate dai project manager. Una volta definito un'insieme di task operativi abbastanza atomici da permetterne lo svolgimento da parte di una persona, questi verranno distribuiti.
Durante lo svolgimento del task, lo sviluppatore al quale è stato assegnato o tutti coloro che in qualche modo vi partecipano, devono tracciare il tempo impiegatovi. Le attribuzioni di unità oraria possono essere principalmente di tipo:

- Sviluppo
- Analisi
- Progettazione
- Documentazione

Questo metodo di Time tracking aiuta il project manager, che può confrontare l'avanzamento con le stime, e valutare l'efficienza degli sviluppatori.

1.3 Strumenti e tecnologie utilizzati

Gli ambienti di lavoro sono molto vari, a seconda delle preferenze e delle necessità. Per i desktop e laptop, è possibile usufruire dei sistemi operativi:

- **Microsoft Windows** attualmente nella versione 7 e 10, ricopre la maggior parte delle utenze sopratutto per l'alta diffusione di software dedicati.
- **Ubuntu** versione 14 o versioni superiori;
- **Apple OSX** Utilizzato in particolar modo dal reparto *user experience*.

Per gli ambienti server invece, c'è una netta preferenza nell'utilizzo di distribuzioni linux, come debian, per il costo delle licenze. Venogno però ancora utilizzati dei server windows server per ospitare dei servizi come Active Directory. Gli strumenti aziendali, possono

essere distrubiti in modo trasversale, o ad uno specifico reparto o team. Concentrandosi sugli strumenti a distribuzione trasversale per i reparti, i principali sono:

- **Office365** pacchetto che include l'ultima versione di office;
- **servizi Google** come Drive, Gmail, Docs. Questi sono utilizzati principalmente per la condivisione e la modifica di documentazione testuale;
- **posta elettronica** su servizio Microsoft Exchange.

Appena entrato a far parte di uno dei reparti, all'utente viene consegnato un'account di dominio aziendale (del tipo nome.cognome@thron.com), che permette l'accesso ai servizi precedentemente citati, e l'accesso a tutti gli ambienti di lavoro (se autorizzati). Nel mio caso l'account era strettamente controllato, e per effettuare una modifica alla configurazione del mio pc, dovevo chiedere l'autorizzazione al personale opportuno.

1.3.1 Versionamento

Per il versionamento, l'azienda utilizza il sistema software di controllo di versione distribuita **Git**. Git è un software di versionamento distribuito, pertanto non ha necessità di un server centrallizzato, ma per raccogliere tutto il codice derivante dai vari progetti delle varie aree sviluppo, THRON possiende un propria istanza di gitlab, posizionata in un'ambiente' locale, accessibile solo all'interno della rete aziendale. Grazie a questo strumento, e al suo corretto utilizzo, è possibile ottenere un pieno controllo del codice software.

Figura 1.6: Logo Git - (https://git-scm.com/(2017/05/08))

I benefici che l'azienda ad utilizzare questo strumento di versionamento in accoppiata con GitLab sono molteplici. I principali però si possono riassumenre in:

- **Branch & Merge**: Il versionamento di Git è organizzato in branch. Un branch è un "ramo" di sviluppo, che possiede uno storico del versionamento. Grazie a questo è possibile applicare le modifiche al codice partendo da uno stesso "commit" (punto di salvataggio) in più versioni, per poi scegliere quella più opportuna e stabile dal quale applicare realmente le modifiche ad un'altro branch o a quello principale ("merge").
- **Pull request**: E' una richiesta formale di Merge tra brach. Grazie a questa è possibile avere una documentazione di quali modifiche vengono apportate ai rami

pincipali e in quale momento; Le modifiche possono quindi essere apportate nel momento più opportuno. Ad esempio molte modifiche o bugfix, possono essere applicate attraverso le pull request in un solo momento, per avere un blocco di modifiche da testare. Attualmente questo strumento viene utilizzato dall'operation manager, per eseguire i test prima del Merge ed effettuare quelli di integrazione. In questo momento può esserre effettuata una *code review* per verificare la bontà del codice. In futuro l'azienda punta ad eseguire i test in automatico al all'inserimento delle modifiche nel repository e nell'operazione di merge che segue una pull request.

- **Tag**: Permette di segnalare un punto del versionamento indicandolo mediante un'apposita etichetta.

- **Information Hiding**: Ogni membro di un qualsiasi team di sviluppo, può avere dei propri progetti, o partecipare a quelli del proprio team. E' possibile consentire la visualizzazione e la modifica del codice di un progetto, a solo una parte dei membri della propria organizzazione. Questa caratteristica per un'azienda che vuole avere un controllo ferreo sul codice è d'importanza fondamentale.

- **Disponibilità**:In assenza di connessione con il repository condiviso, è comunque possibile continuare a lavorare al proprio progetto su un clone locale, successivamente, potranno essere caricate le proprie modifiche (operazione chiamata "pull") per renderle visibili ai propri collaboratori

- **Condivisione e cooperazione**: L'utilizzo di un **bucket** centralizzato, e delle pull request, spinge alla cooperazione i colleghi, che possono richiedere una rapida code review o consigli, evasi con semplici operazioni da parte del collaboratore al quale vengono richiesti. Inoltre agevola lo sviluppo e il miglioramento di alcuni progetti di interesse comune a più servizi o team.

Per l'utilizzo dello stumento non sono fornite interfacce grafiche, ma il solo pacchetto ufficiale di Git da bash. Spesso però viene utilizzato il pacchetto di versionamento per Git fornito da vari Ide come Intellj IDEA o PyCharm.

1.3.2 Sviluppo

L'azienda, avendo un prodotto molto complesso che comprende molti ambiti dell'informatica e molto differenti tra loro, non possiede un'unica suite di strumenti comune a tutti i team di sviluppo. Per questo motivo, ogni team sceglie autonomamente gli strumenti più opportuni per i processi riguardanti lo sviluppo e i test.
Per gli IDE il principio è lo stesso: in base ai linguaggi utilizzati ogni team valuta quelli che ritiene migliori e desidera utilizzare. Se lo strumento è open source, anche una singola persona può provvedere a installarlo ed utilizzarlo; se invece è a pagamento, deve essere effettuata una richiesta formale.

Mentre framework e strumenti sono di libera scelta, i linguaggi alla base del prodotto sono:

- Javascript: Utilizzato in abbinata ad angular 1.x per lo sviluppo della parte front-end
- Scala: utilizzato per lo sviluppo della parte back-end e parte dell'intelligence, per le sue pontezialità e la portabilità (essendo eseguito sulla jvm)
- Python: Utilizzato per lo sviluppo di alcuni scritp di analisi e conversione.

Per la documentazione, fino a poco tempo prima del mio arrivo, era sufficiente produrre un PDF, con le caratteristiche del servizio e, nel caso fosse presente un'interfaccia API, l'apposita documentazione swagger in formato YAML.
Durante il mio stage però, si stava testando la possibilità di documentare i servizi mediante il formato markdown, supportato anche da Gitlab, per accorpare la documentazione nella repository del codice, e versionare anche quest'ultima.
La documentazione del codice avviene con l'utilizzo di appositi commenti, leggibili dagli IDE come IntelliJIDEA e PyCharm, maggiormanete utilizzati perchè sono disponibili licenze e sono specializzati linguaggi utilizzati.

Figura 1.7: Aggiorni IDE utilizzati

Per effettuare chiamate HTTP, e per testare il funzionamento dei servizi REST, lo strumento utilizzato è POSTMAN. Questo software permette di eseguire in modo rapido e tramite una comoda interfaccia grafica, tutte le operazioni basate su protocollo http. Il test di un'API può essere fatto anche in modo automatico, con un report delle chiamate eseguite. POSTMAN è Freeweare nella sua versione base.

1.3.3 Supporto ai processi organizzativi

Per i processi organizzativi si utilizza, al contrario dell'ambito sviluppo, uno strumento unico per tutti i team. Questo permette un controllo centralizzato da parte della direzione del reparto, su un'unico strumento.
Youtrack è lo strumento utilizzato per la creazione e assegnazione di attività nominate "Issue". Questo software permette un'organizzazione a progetti delle attività aziendali e supporta anche alcune delle caratteristiche del modello di sviluppo agile. Alla creazione

di un'issue oltre a un sommario, una descrizione e una serie di file, è possibile anche assegnarvi:

- **Priorità**: indica il livello di urgenza di quell'attività;
- **Stato**: indica l'avanzamento di quell'attività;
- **Assegnatario**: quest'attributo può esserci, o può essere non ancora assegnato (scrum board), ed indica chi ha l'onere di svolgere quell'attività.
- **Tipo**: Indica di che tipo di Issue si tratta, ed esempio "Bug Fix" o "Feature".
- **Tempo di esecuzione previsto**: Indica quanto tempo è previsto per il completamento di quell'issue.

Youtrack offre anche la possibilità di descrivere come è stato speso il tempo in quell'issue, con unità oraria o giornaliera, proponendo dei report sulla base di quest'ultima informazione.

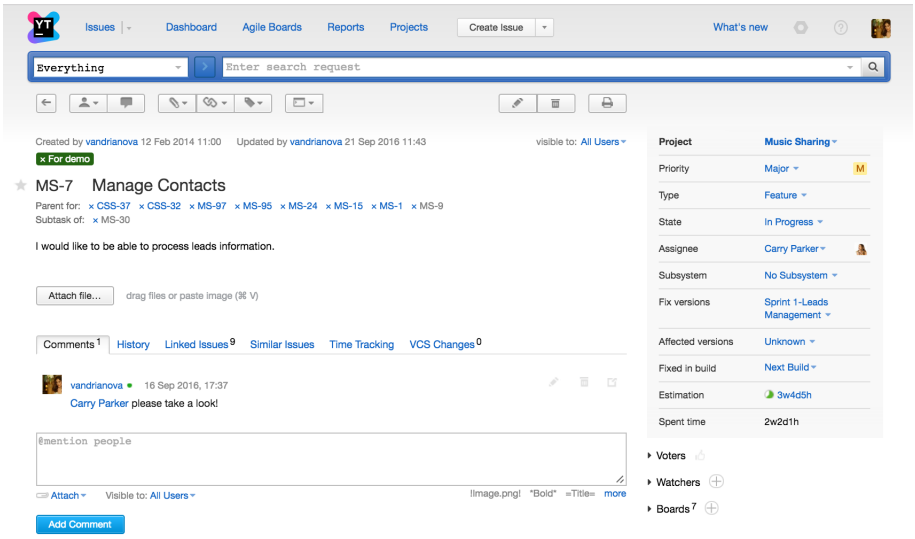


Figura 1.8: Jetbrains YouTrack

E' anche possibile commentare l'issue ed aggiungere degli allegati sui propri commenti. Questo aiuta a rendere tracciabli le informazioni richieste e le decisioni prese. YouTrack permette inoltre di allacciare le informazioni di un brach git ad un'issue, per mostrare i progressi nello sviluppo, di pari passo alla segnalazione delle ore impiegate attraverso il time tracking.

Capitolo 2

Lo stage per l'azienda ospitante

2.1 Necessità aziendali

L'importanza della ricerca Possedere una ricerca funzionale, in un prodotto che ha come obiettivo principale l'accentramento di contenuti multimediali, è importante per semplificarne l'utilizzo. Oltre a dover essere una ricerca efficace, deve essere anche efficiente, per l'elevato numero di contenuti della piattaforma. Le entità che necessitano di una funzionalità di ricerca sono:

- Utenti;
- Contenuti;
- Tag;
- Chat;
- Metadati.

Ognuno di queste entità, memorizzate in un database non relazionale in formato json, ha struttura e proprietà differenti.
La ricerca essendo disponibile ad ogni utente, deve "trovare" solo le entità disponibili all'utente che la effettua. Questo deve avvenire mediante l'utilizzo di apposite proprietà chiamate ACL. Ne esistono di due tipi, dirette o ereditate dal contesto. Le ACL possono essere assegnate ad un'utente o a un gruppo.
Con l'aumento dei contenuti, e le esigenze dei clienti che continuano a crescere, è divenuto necessario per l'azienda considerare l'opzione di un refactoring del servizio di ricerca. Per capire quali fossero le funzionalità che il nuovo servizio avrebbe dovuto offrire oltre ad una maggiore efficienza, si è svolto un'approfondito studio di usabilità.

Studio di usabilità Uno dei punti d'interesse dell'azienda, era la possibilità di poter unire le varie ricerche, che ora sono disponibili in pagine diverse, in un'unico punto per

tutte le entità. Dopo un'attenta analisi però, quest'ultima risulta un'opzione non praticabile a meno di una ristrutturazione dell'intera bacheca di THRON, fornendo una pagina solo per i risultati di ricerca, nel quale potessero essere combinati i diversi tipi di oggetto. Infatti data la natura promiscua degli elementi da ricercare, non sarebbe possibile utilizzare una proprietà di una singola entità come criterio se non prevista a priori. L'opzione migliore è quindi slegare il servizio dall'usabilità, fornendone uno per entità, ma con la stesso metodo di utilizzo, in modo da poter eventualmente unire i risultati in futuro.
Ogni una delle entità precedentemente descritte ha necessità di ricerca differenti. In particolare i contenuti, essendo il punto d'interesse principale del prodotto, necessita di 2 modalità:

- Ricerca veloce: filtri per contesto (tag) e ricerca testuale su alcuni attributi;
- Ricerca avanzata: filtri avanzati come data, metadati e selezione dei campi nel quale effettuare la ricerca testuale.

La ricerca veloce deve proporre un'autocompletamento, bastato su testo, tag e metadati, mostrando in quali attributi è stato ritrovato il risultato.
Successivamente grazie all'autocompletamento deve essere possibile inserire il risutato selezionato come filtro della ricerca (questo verrà effettuato client/side) ed eseguire il processo di ricerca o reiterare quello di autocompletamento. Questa funzione è già parzialmente disponibile. La possibilità di inserire un tag come filtro sulla ricerca veloce, è già disponibile su THRON, ma l'autocompletamento non mostra anche i risultati della ricerca testuale, bensì solo quella che riguarda i tag.
Una volta terminato l'inserimento degli attributi, al click del pulsante "cerca" o di uno dei suggerimenti, vengono mostrati i risultati di ricerca.

2.1.1 Richieste dei clienti

La necessità di questo refactoring della ricerca, è nata principalmente da alcune richieste avanzate dalla clientela.
Alcuni clienti affermano che in presenza di una grande quantità di dati e tag, risulta difficile trovare i contenuti, questo perchè la ricerca non risulta abbastanza efficace.
Il numero di campi ricercabili non è ristretto e sia con la ricerca semplice che in quella avanzata, risulta difficile capire quale sia l'attributo che ha portato quell'entità ad essere inserita tra i risultati, dato che non è segnalato in nessun modo. Questa funzionalità è comunemente conosciuta come highlighting, ed è la capacità di una ricerca di mostrare dove e cosa ha trovato il servizio.
Prendendo sempre come caso specifico i contenuti, l'autocompletamento viene fornito solo per i tag, e non per il titolo o la descrizione degli stessi. Questo è utile solo in parte. Analizzando due casi, è subito evidente come l'autocompletamento o il Fuzzy match su quei campi, se fatto in modo efficiente, sarebbe di maggiore utilità.

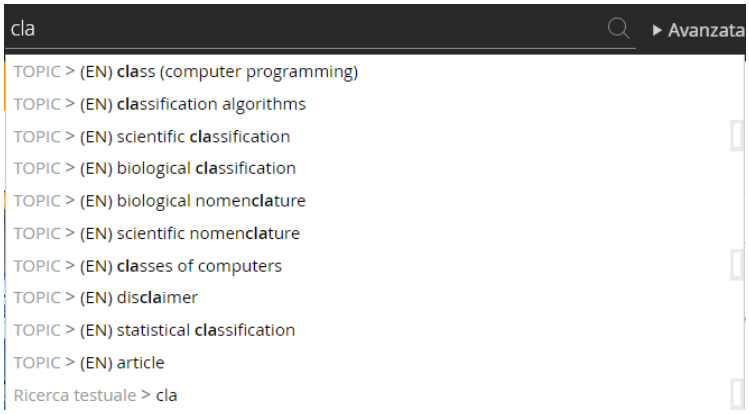


Figura 2.1: Attuale autocompletamento - THRON

Dopo una sintesi e un'analisi approfondita, le richieste dei clienti a livello di usabilità si possono interpretare principalmente in due esigenze:

- Offrire una maggiore chiarezza durante l'autocompletamento, potendo ridurre maggiormente il numero dei risultati, attraverso una specifica maggiore sui campi di ricerca;
- Visualizzare per quale attributo l'oggetto è rientrato tra i risultati della ricerca.

L'azienda necessita quindi di servizi di ricerca, in grado di offrire al front-end queste funzionalità in modo efficiente, e con un'interfaccia REST simile per tutte le entità.

2.1.2 Limiti dell'attuale servizio

Già nella sezione precedente, ho introdotto alcuni dei limiti e problematiche che ha l'attuale servizio di ricerca fornito da THRON. Prima di descrivere a cosa sono dovute quest'ultime, è necessario spiegare su quali tecnologie è basata la ricerca attuale.
L'attuale ricerca è basata su delle query nel database principale di THRON, che è un'istanza di MongoDB. Senza scendere troppo nel dettaglio di questa tecnologia, MongoDB è un DBMS non relazionale molto efficiente nel salvataggio e reperimento dei documenti JSON, ma non altrettanto per ricerche Full-text o ricerche su elementi non indicizzati. Di fatto le ricerche su MongoDB si basano su degli indici nei quali sono organizzati i documenti. Da questo ne deriva l'immediato limite, che il numero di campi nei quali è possibile effettuare una ricerca efficiente, è direttamente correlato al numero di indici che MongoDB può supportare all'interno di una collezione di elementi.
MongoDB non possiede operazioni su stringhe differenti dall'operatore Like e Regex. Ne deriva che tutta la parte di ricerca full-text deve essere gestita dal servizio, generando accuratamente le query. Altra conseguenza dell'utilizzo di questo strumento è che la ricerca Fuzzy ovvero con errore non è supportata, sarebbe pertanto necessario effettuare più chiamate consecutive al database per ottenere un risultato.
Un limite che invece deriva da un vincolo strutturale e non tecnologico è che tutti i ser-

vizi di ricerca non hanno un'interfaccia comune di interrogazione, senza la possibilità di costruire una logica booleana di scelte.
Altra difficoltà del servizio di ricerca attuale, quando si inserisce una stringa, ogni parola viene considerata separatamente, e nel documento risultante, devono essere tutte contenute ma nello stesso ordine di inserimento, anche se possono essere separate da parole. L'attuale servizio inoltre, nel caso di una ricerca testuale, non prevede nessun criterio di ordinamento che distingua quali risultati hanno maggior rilevanza, sulla base del risultato cercato. In particolare, supponendo di voler cercare "ciao pippo", non ci sarebbe nessuna differenza tra un documento che contiene:

- "ciao pippo, giovanni" nel titolo;
- "ciao sono pippo, chiama giovanni" nel titolo o nella descrizione;
- Un tag chiamato "dì ciao a pippo".

La modalità di ordinamento nel caso ci fossero risultati, si basa su alcuni campi degli attributi, e non sulla rilevanza del documento in base alla ricerca. Questo tipo di funzionalità può essere sintetizzato con il concetto di Ranking di un documento.

2.2 Obiettivi dello stage

Ciò che ha spinto l'azienda a proporre questo stage, sono state alcune richieste dei clienti.
Anche la mancanza di performance del servizio di ricerca attuale con l'aumentare dei dati e l'impossibilità di soddisfare alcune delle esigenze che il mercato richiedeva, a causa dell'architettura attuale del servizio e delle tecnologie, impiegate ha spinto THRON a pensare di riprogettare l'architettura del servizio.
Prima di svolgere un'operazione così costosa, e bloccare altri sviluppi, è necessario valutarne i benefici, se tutti i requisiti siano soddifabili, a che costi e con quali strumenti.
Il poter utilizzare una risorsa esterna, proveniente dall'università, permette all'azienda di valutare il futuro sviluppo, senza che i propri dipendenti sospendano il proprio lavoro. Questo le da anche l'opportunità di valutare nuove persone che potrebbero entrare a far parte del team aziendale.
Lo stage che l'azienda propone ha quindi come obbietivi quelli descritti nelle seguente tabella.

Tabella 2.1: Soddisfacimento obiettivi stage

Identificativo	Descrizione
O01	Definizione DB e schema per registrazioni metadati contenuti.
O02	Definizione struttura servizi per le ricerche (query language, parametri ricerca, struttura oggetti di risposta del servizio)
O03	AWS Elastic Search: importazione dati da Database Mongodb. Definizione struttura collezioni ed indici, performace test su dati destrutturati e su dataset da 10 milioni di documenti.
O04	Sistema di syncronizzazione tra main repository (Mongodb) dei contenuti ed AWS Elastic Search.
O05	Definizione test search webservice in AWS: API Gateway + Lamba Function+ ElasticSearch (linguaggio Scala).

2.3 Vincoli

In accordo con l'azienda, lo svolgimento dello stage si è tenuto presso la sede in Piazzola sul Brenta (PD). Questo ha consentito un dialogo continuo tra stagista e membri dell'azienda. Darà l'opportunità allo studente di apprendere nozioni da programmatori di più lunga esperienza, potendo questi facilmente chiedere consiglio al proprio tutor per consigli, su architettura e interfacce.
Durante l'intera durata dello stage, ho lavorato per portare a termine gli obiettivi prefissati precedentemente, e ad ogni attività conclusa, ho illustrutato al tutor i risultati e discusso la successiva attività da svolgere. Essendo il mio tutor una figura molto impegnata, mi è stata affidata una figura di supporto, con la quale dialogare e chiedere pareri e consigli. A quest'ultimo quindi, una volta terminata un parte di prodotto, dovevo mostrare i risultati, e raccogliere i suoi suggerimenti migliorie o errori a cui porre rimedio. L'intero lavoro svolto in azienda, doveva rispettare le norme di progettazione e codifica attualmente in uso. Il lavoro doveva essere accuratamente versionato e documentato sullo strumento di versionamento aziendale.
Nel caso avessi ritenuto opportuno l'utilizzo di servizi provenienti da cloud Amazon, dovevo discuterne costi e benefici e caso d'uso, con una figura preposta alla funzione, che ne gestisce permessi ed utilizzo. Dopo la sua approvazione e quella del tutor, mi venivano consegnate le chiavi di accesso a quello specifico servizio.
Il monitoraggio delle mie attività e dei risultati raggiunti, veniva garantita attraverso riunioni, durante le quali discutere delle nozioni apprese e verificare se i limiti degli strumenti

utilizzati erano accettabili o fosse necessario un cambio tecnologico.
Al termine dello stage era prevista una presentazione su tutto il lavoro svolto e i risultati ottenuti, al resto del team.

2.3.1 Temporali

Per il completamento dello stage, l'università ha imposto lo svolgimento di un minimo di 300 ore effettive di lavoro. Inizialmente lo stage, era stato calibrato su 320 ore di lavoro per permette lo svolgimento di tutte le attività senza inficiare sulla loro qualità. Le ore sono state distribuite quindi in 8 settimane lavorative di 40 ore a settimana seguendo l'orario aziendale. L'orario aziendale era articolato su 5 giorni settimanali

- 8:30 - 12:30
- 14:00 - 18:00.

Le attività previste all'interno dello stage inizialmente sono state suddivise nelle modalità descritte nella seguente tabella. Quest'ultime hanno coperto tutti gli obiettivi previsti.

Tabella 2.2: Attività preveiste

68	Definizione dell'architettura per il motore di ricerca
32	*Definizione DB e schema per registrazione metadati dei contenuti*
36	*Definizione struttura servizi per le ricerche (query language, parametri ricerca, struttura oggetti di risposta del servizio)*
252	Proof-of-Concept per testare l'architettura utilizzando AWS Elastic Search
6	*AWS Elastic Search: definizione struttura collezioni ed indici*
10	*AWS Elastic Search: importazione dati da Database MongoDB*
48	*AWS Elastic Search: test performance e limitazioni su funzionalità di ranking e ordinamento, eseguiti su dati destrutturati e su dataset da 10 milioni di documenti*
80	*Sistema di sincronizzazione tra main repository (MongoDB) dei contenuti ed AWS Elasticsearch*
8	*Testing sistema di sincronizzazione*
60	*Definizione Lambda Function per il test search*
8	*Definizione API Gataway per il test search*
32	*Definizione dei test per il test search*

Queste attività sono state distribuite nelle 8 settimane in modalità sostanzialmente sequenziale rispetto all'ordine della tabella sopra descritta.
Come ci si poteva aspettare da un progetto sperimentale, la suddivisione oraria è stata

sensibilmente differente da quella sopra riportata. Nello specifico, la codifica del sistema di sincronizzazione tra main repository ed Elasticsearch è stata sensibilmente più veloce rispetto alle aspettative.

2.3.2 Tecnologici

Per la POC, era specificatamente richiesto l'utilizzo di Elasticsearch service, presente tra i servizi di AWS. Il motivo di scegliere "Elasticsearch as a service" risiede nella scalabilità immediata e senza vincoli, nel quasi nullo costo di manutenzione, e nella possibiltà di calcolare precisamente il costo netto del mantenimento, cosa molto più difficile nel caso dell'istallazione dello strumento in un'apposita macchina , se pur open source. Il servizio di ricerca del Proof of concept invece, avrebbe doto essere eseguito sul servizio Lambda di Amazon web service. Questo permette di avere un microservizio sempre disponibile ed efficiente. Nel caso di un servizio POC, che riceve poche chiamate in un lungo lasso di tempo, la fatturazione a numero di esecuzioni rende Lambda un'ottima tecnologia per il caso d'uso. Per l'impiego di altre tecnologie e linguaggi durante lo stage, non c'era una vera e propria imposizione. Essendo però l'azienda oriendata sui servizi cloud di Amazon AWS, mi è stato richiesto di utilizzare quelli se validi per il caso d'uso.
L'azienda ha espresso comuqnue delle preferenze sui linguaggi da utilizzare durante lo sviluppo, perchè sono quelli maggiormente utilizzati da loro. Questi linguaggi sono Scala e Python. Il loro impiego, è dovuto a diversi motivi:

- Sono entrambi multipiattaforma;
- C'è un'alta diffusione di librerie fortemente supportate, specialmente nel caso di Python.
- Le potenzialità e la flessibilità che entrambi i linguaggi posseggono, attributi predominanti nel linguaggio Scala. Essi infatti possono essere usati in ambiti anche molto differenti.

Grazie alla presenza di una libreria molto semplice e supportata per lo sviluppo e il deploy di servizi REST in AWS Lambda, è stato scelto l'utilizzo di Python 2.7 per il servizio della POC, che verrà eseguito su AWS Lambda. Per la gestione dell'interfaccia REST del servizio di ricerca utilizzata per collegarsi a Lambda, è stato invece scelto in accordo con l'azienda l'utilizzo di API Gataway. Per invece quanto riguarda la sincronizzazione tra repository, non è stato prefissato alcun vincolo tenologico, se non la possibilità di inglobare quest'ultimo in un container di Docker.

Figura 2.2: Tecnologie POC

2.4 Motivazioni personali di scelta stage

Prima di questo stage, avevo già affrontato altre esperienze lavorative, e ho sempre considerato l'esperienza sul campo come un'ottimo metodo per approfondire le proprie conoscenze. Durante gli anni di università,nei corsi che ho affrontato, mi sono imbattuto spesso in nozioni che già avevo appreso per motivi lavorativi. Viceversa, proseguendo con l'università, mi accorgevo che alcuni lavori fattti in precedenza non erano validi quanto pensassi. Le mie aspettative riguardandi questo stage quindi erano di riuscire ad approfondire le conoscenze fin'ora apprese, e di aquisirne altre.
Per poter aumentare le mie possibilità di scelta, e capire quali fossero le attuali tendenze di sviluppo, ho deciso di partecipare all'evento STAGE-IT organizzato dall'università di Padova. Durante quell'occasione ho potuto presentarmi a diverse aziende, e discutere delle eventuali opportunità di stage. Quell'ocasione è stata anche utile per capire come presentarsi alle aziende, e cosa fosse importante nel mondo del lavoro al quale mi sto per affacciare.
Inizialmente i requisiti che ricercavo dal mio futuro stage erano:

- Lavorare nella parte server-side di un prodotto o progetto, e poter sviluppare maggiori conoscenze sulla progettazione e sui metodi si sviluppo di tale porzione di architettura;
- Poter raffinare le mie conoscenze riguardanti la comunicazione tra web-app e i loro relativi servizi;
- Poter aumentare le mie conoscenze riguardanti l'utilizzo dei servizi Amazon;
- Poter sfruttare almeno in parte alcune delle mie conoscenze pregresse e poterle mettere in discussione;

Se non avevessi trovato nessun offerta in grado di darmi ciò, avrei optato per il settore totalmente diverso della consulenza.
Precedentemente all'evento ci era stata già fornita una lista di aziende partecipanti, e proprio in quell'occasione avevo puntato ad un progetto offerto appunto da THRON per

lo sviluppo di un servizio di chatting. Successivamente, sono stato ricontattato per un colloquio, ma a mia sorpresa mi è stato offerto un'altro stage, riguardante per l'appunto un servizio di ricerca. Questo, considerato che ricopriva tutti i punti sopra descritti, è stata la scelta per me più opportuna.

Capitolo 3

Svolgimento dello stage

3.1 Pianificazione delle attività

Nel periodo antecedente l'inizio dello stage, dopo la proposta durante il colloquio, il *Chief operating officer* (COO) di THRON mi ha fornito un insieme di macro attività coincidenti con gli obbiettivi dello stage, che avrei dovuto svolgere durante i due mesi di stage.
Ho provveduto in seguito a redigere un piano di lavoro, in collaborazione con i due tutor (interno e aziendale), che avesse delle attività meglio definite e più parallelizzabili. Ho già riportato il dettaglio di tali attività nella tabella 2.2 presente alla sezione 2.3.1 Dopo il completamento di ogni attività, abbiamo stabilito inoltre che avrei dovuto fare un piccolo resoconto verbale a lui e al mio referente di ciò che avevo fatto.
Durante l'attività di studio invece, avrei dovuto redigere un documento, contente un riassunto delle considerazioni e dei dettagli rilavanti durante lo studio. A seguito di questi incontri, veniva scelta la successiva attività da svolgere, che poteva essere un'iterazione della precedente, o un'attività differente.
Al completamento dello stage, ho dovuto preparare una piccola presentazione, supportata dall'utilizzo di apposite slide, da mostrare al mio tutor e al resto del team di sviluppo. All'interno di questa vi è stato spiegato tutto il lavoro svolto, gli accorgimenti, i risultati ottenuti e le difficoltà incontrate.

3.2 Studio di Elasticsearch

3.2.1 Caratteristiche

La prima parte del mio stage, prevedeva lo studio delle caratteristiche e dei limiti di Elasticsearch, per appurare se fosse lo strumento adatto alle esigenze, e potesse quindi soddisfare tutti i requisiti.
Elasticsearch è una tecnologia open source divenuta nel tempo molto popolare nell'ambito Big Data, sia negli ambienti *enterprise* che nel settore del *cloud computing* per la capacità di ricercare, analizzare e mostrare dati contenuti nei documenti in formato

JSON.
Il formato JSON (acronimo di JavaScript Object Notation) è un formato open standard che utilizza testo in linguaggio di alto livello comprensibile all'uomo per trasmettere oggetti di dati che consistano nella coppia attributo-valore.
ElasticSearch si propone come database di ricerca basato su Apache Lucene, una libreria open source per il recupero delle informazioni, le cui caratteristiche peculiari vengono rese disponibili da Elasticsearch agli utenti tramite API da interrogare mediante HTTP. Per la sua conformazione, che verrà descritta in seguito, Elasticsearch è scalabile nella ricerca fino anche a petabyte di dati. Questo però vale solo se all'aumentare dei dati ricercati, il cluster messo a disposizione per questa ricerca, viene mantenuto al passo con il carico esercitato.
I concetti chiave di Elasticsearch si possono riassumere nei seguenti termini:

- **Node**: Si riferisce a una singola istanza in esecuzione di elasticsearch. Singolo server fisici e virtuali può ospitare più nodi a seconda delle capacità dei loro risorse fisiche, come la RAM, storage e potenza di elaborazione.

- **Cluster**: Si tratta di una raccolta di uno o più nodi. Cluster offre collettivi capacità di indicizzazione e di ricerca in tutti i nodi di tutti i dati.

- **Index**: Si tratta di una raccolta di diversi tipi di documenti e le proprietà del documento. Indice utilizza anche il concetto di frammenti per migliorare le prestazioni. Ad esempio, un insieme di documento contiene dati di un'applicazione di social networking.

- **Type/Mapping**: Si tratta di una raccolta di documenti che condividono un insieme di campi comuni presenti nello stesso indice. Ad esempio, un indice contiene i dati di un'applicazione di social networking, e quindi non ci può essere un tipo specifico per i dati del profilo utente, un altro tipo di messaggistica e dati di un altro per i dati commenti. Definendo una mappa per un indice, ad un attributo è possibile definirne oltre che il tipo, anche caratteristiche di analisi o indicizzazzione.

- **Document**: Si tratta di un insieme di campi in maniera specifica definita in formato JSON. Ogni documento appartiene a un tipo e risiede all'interno di un indice. Ogni documento è associato un identificatore univoco, chiamato UID.

- **Shard**: Gli indici sono in orizzontale suddivisi in frammenti. Ciò significa che ogni frammento contiene tutte le proprietà di documento, ma contiene meno oggetti JSON di quanti ne contenga l'indice. La separazione orizzontale rende *Shard* un nodo indipendente, che può essere utilizzato in ogni nodo. Il rammento primario è la parte orizzontale originale di un indice e quindi questi frammenti primari vengono replicati in frammenti di replica.

- **Replicas**: Elasticsearch consente all'utente di creare repliche dei loro indici e schegge. Replication aiuta non solo ad aumentare la disponibilità di dati in caso di

fallimento, ma migliora anche le prestazioni di ricerca per eseguire un'operazione di ricerca in parallelo in queste repliche.

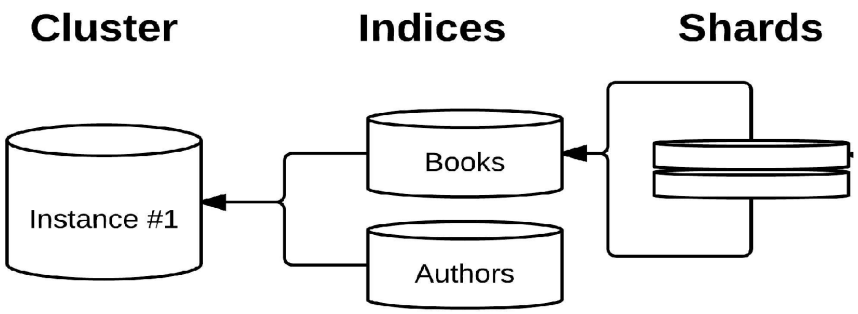


Figura 3.1: Organizzazione alto livello Elasticsearch

Ogni indice (index) di Elasticsearch, prevede su ogni tipo una mappa (mapping). Quest'ultima descrive le proprietà comuni agli oggetti, attraverso l'assegnazione di un tipo. Questi ultimi sono molti, ma i più utilizzati sono: *date, text, keyword, number, array* e *nested object*.
Una caratteristica molto importante di Elastisearch è legata agli attributi di tipo testo (text). A questi è possibile infatti assegnare nella mappa del tipo due proprietà chiamate *analyzer* e *tokenizer*. L'analyzer è una modalità di ricerca nel testo, e prevede nella sua definizione anche un tokenizer per la sezione del testo in più sottostringhe da indicizzare. Un analizer viene utilizzato per l'indicizzazione di un documento una volta inserito, mentre l'altro per il testo che verrà confrontato durante la ricerca.
Il tokenizer è il metodo di sezione del testo in *Token* ovvero pezzi più piccoli. Ce ne sono molti a disposizione, e a seconda della tipologia, possono essere personalizzati con delle specifiche proprietà. Questi possono essere divisi in più categorie, dove le principali sono:

- **Structured Text Tokenizer**: Suddividono particolari tipi di testo strutturato, come path o elementi che seguano un pattern che può essere espresso tramite le espressioni regolari.
- **Partial Word Tokenizer**: Separano il testo in porzioni di testo, ignorando anche elementi definiti come punteggiatura e spazi.
- **Word Oriented Tokenizer**: Suddividono il testo in singole parole.

Al risultato di questi dopo si possono applicare anche diversi *token filters*, i più comuni sono: *lowercase* che trasforma tutte le lettere in minuscolo e *stopword* che rimuove tutta la punteggiatura.
La ricerca full text di Elasticsearch si basa proprio su questi token, difatti Lucene esegue una ricerca di *pattern matching* tra il risultato del testo sottoposto ad analisi e i token organizzati nello specifico indice.

Una caratteristica di elasticsearch utile nel caso di ricerche consecutivi sugli stessi elementi è la **Cache** di shard. Questa permette di memorizzare alcuni dei filtri eseguiti nelle ricerche più recenti per velocizzare le ricerche effettuate con i medesimi filtri.

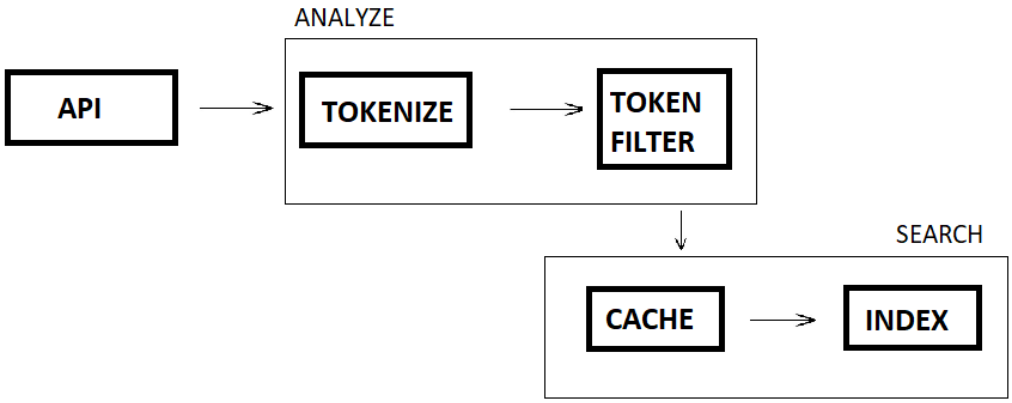


Figura 3.2: Flusso di ricerca con analisi

3.2.2 Limiti e punti critici

Elasticsearch però ha alcuni limiti e condizioni, che gli vengono imposte non solo dal fatto di essere organizzato in molte istanze differenti di Lucene, ma alcune derivano proprio dalle capacità di questa libreria. L'introduzione di questi, servirà a motivare alcune delle scelte intraprese durante lo svolgimento dello stage, da parte mia e del tutor aziendale.

- **Campi di ricerca**: se un campo non è presente nella mappa sopracitata, anche se è presente negli oggetti, non potrà in alcun modo essere cercato, perché non indicizzato;
- **Indicizzazione**: se un numero elevato di elementi è appena stato inserito, l'indicizzazione, ovvero l'ordinamento degli elementi all'interno di un indice, potrebbe essere molto lunga; Questo fa calare di molto le prestazioni in quel periodo di tempo e non permette di ricercare i documenti appena inseriti;
- **Nested object in array**: Un sott'oggetto presente in un array, se non viene inserito come "nested object" nella mappa, l'oggetto viene scomposto ed i suoi campi vengono inseriti in più array che contengono un determinato campo di tutti gli oggetti. Questa particolarità è dovuta a *Lucene*, che non supporta gli array di oggetti. In questo caso non è possibile mantenere la correlazione tra campi di un singolo oggetto in un array.

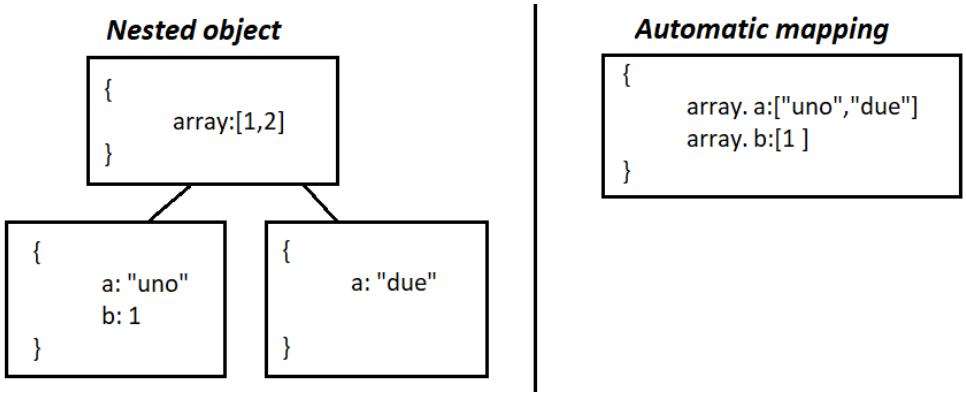


Figura 3.3: Esemplificazione punto critico nested object

Dall'altro lato però, effettuare la ricerca su array contenti *nested object*, la rende molto più lenta. Ove possibile è quindi ricercare un'altra soluzione;

Oltre ai limiti sopra descritti, Elasticsearch ha delle altre peculiarità che bisogna conoscere prima di utilizzare alcune funzioni.
Una di queste riguarda l'uso dei *limit* con gli ordinamenti. Difatti se si richiede un ordinamento con un determinato limite di risultati, Elasticsearch ordinerà i risultati dei singoli *shard*, poi preleverà in primi N elementi richiesti come *limit* e li ordinerà a sua volta. Non è però detto che il risultato coinciderà realmente con i primi N elementi secondo l'ordinamento totale.

3.3 Sviluppo del servizio per il POC

3.3.1 Requisiti di ricerca

Prima di iniziare lo studio di Elasticsearch e le seguenti attività di progettazione e codifica del POC, mi sono stati consegnati dei documenti, illustranti quelli che erano i requisiti base che il servizio doveva possedere.
Questi prevedevano i primo luogo le entità ricercabili nel futuro servizio di ricerca. Essendo molte, in accordo con il tutor aziendale, abbiamo stabilito che il POC si sarebbe basato su un numero ristretto di entità. In particolare sono state scelte quelle più complesse e che nell'ambiente di produzione erano numericamente più rilevanti:

- Contenuti (contents)
- Utenti (users)
- Tag (Tags)
- Cartelle (Category)

I campi in cui questi dovevano essere ricercati erano differenti per ogni entità.
Ho prestato particolare attenzione alle ACL, ovvero i permessi di visualizzazione. Questi

imponevano semplicemente di distinguere chi è autorizzato alla visualizzazione dell'entità e chi no. Nel particolare caso dei contenuti però, è stato richiesto anche di riuscire a distinguere quali fossero le ACL assegnate direttamente a quell'utente, e quali invece fossero quelle ereditate dal contesto(per esempio cartelle). Gli altri requisiti riguardano invece le funzionalità che il servizio avrebbe dovuto soddisfare. Quest'ultimi erano il risultato di un'analisi delle esigenze dell'utente quando si sarebbe approcciato ad effettuare la ricerca. Altro importate vincolo consisteva nella possibilità di restringere il numero di lingue nel quale effettuare la ricerca testuale, all'interno degli attibuti locali di entrambe le proprietà. I principali requisiti possono essere riassunti nella tabella sottostante, dove la numerazione affiancata è quella che ho utilizzato per la verifica.

Tabella 3.1: Requisiti funzionali

Identificativo	**Descrizione**
1	Ricercabilità nei campi prestabiliti di ogni entità.
2	Ricerca eseguita in meno di un secondo con cluster minimo fuori carico.
3	La ricerca deve essere effettuata con sempre filtro **clientId**.
4	Offerta di paginazione offset-limit.
5	La ricerca full-text effettuata con parole in AND in ordine uguale o sparso.
6	La ricerca full-text effettuata con parole in OR.
7	La ricerca full-text effettuata con match esatto del testo.
8	La ricerca full-text non deve essere *case sensitive*.
9	Highlight nei risultati della ricerca full-text.
10	Deve essere disponibile una funzione di autocompletamento.
11	Per le proprietà che lo prevedono deve essere possibile selezionare la lingua nel quale eseguire la ricerca full-text.
12	La stringa ricercata deve essere valutata anche nei grammi dei testi indicizzati.

Particolare attenzione va fatta al requisito 5. Questo difatti è quello che maggiormente giustifica il lavoro effettuato durante lo stage, perchè è una delle maggiori potenzialità offerte dal full-test search. L'attuale servizio prevede la ricerca testuale in and, separando la stringa in una forma simile all'edge ngram, ma le parole devono essere nello stesso ordine nel quale vengono inserite, anche se separate da altre. Con "ordine sparso" si indente la possibilità di trovare un risultato anche con inversione delle parole inserite. Si può esprimere meglio questo tipo di problematica che il servizio deve risolvere con il requisito 5 nello schema sottostante.

Figura 3.4: "Ordine sparso" full-text esemplificazione

3.3.2 Design degli indici Elasticsearch

Il design degli indici per Elasticsearch (index), è stato l'operazione fulcro della POC. L'indicizzazione automatica, senza uno schema definito, non permette difatti una ricerca full text che rispettasse i requisiti stabiliti. Affidarsi ad Elasticsearch per la definizione degli indici, comporta anche una perdita dei sott'oggetti come entità. Le loro proprietà, verrebbero spostate all'interno dell'oggetto principale, e cio comporterebbe un formato inaspettato del documento indicizzato. Essendo lo studio, effettuato avendo le sole nozioni teoriche, sono state apportate in seguito modifiche sulla base dei test di performance eseguiti in seguito al'implementazione.
Le scelte principali che sono state effettuate durante il design degli indici, riguardano il formato degli attributi e la *tokenizzazion*, ovvero il formato nel quale effettuare la tokenizzazione. Per la creazione degli indici ho proceduto in modo quasi sequenziale, partendo dai contenuti e poi proseguendo la definizione anche per le altre entità.
Come prima operazione , ho classificato quali proprietà presenti nei documenti del main repository, fossero necessarie per le ricerche che avrei dovuto affrontare secondo i requisiti. Successivamente ho deciso come strutturare i dati nel nuovo database, e solo poi ho effettuato una sperimentazione sui vari tipi di tokenizzazione valutandone, le caratteristiche e la funzionalità. Quest'ultima operazione è stata effettuata attraverso la strutturazione e il test delle query che sarebbero state poi utilizzate nell'interrogazione.
La principale difficoltà affrontata nel strutturare i dati, sono state le proprietà, che nel *main repostory* sono rappresentate come array di oggetti.
Come già anticipato in 3.2 il modellare degli array all'interno della mappa, consente ad Elasticsearch di considerare un elemento di tali come un puntatore ad un'oggetto indicizzato altrove. Questo però ne diminuisce nettamente le prestazioni, mentre per i sott'oggetti questo avviene in modo molto meno influente. Nel caso di alcune proprietà come quella relativa alle ACL che, denominata "#inverseAclRoules", che presentava questo problema, ho semplicemente eliminato tutte le informazioni aggiuntive, come permessi di scrittura o amministrazione, rendendo un array di oggetti, uno che avesse semplicemente una keyword, che rappresenta l'utente o il gruppo che ha diritto di accesso. Per

mantenere il contesto del permesso, come richiesto da requisito, sono stati utilizzati due array, uno che contiene i permessi diretti, e uno che contenesse quelli inversi. In questo modo è possibile soddisfare tutti i requisiti di autorizzazione, effettuando un ricerca dell'identificativo dell'utente o dei gruppi al quale appartiene.
Una proprietà in particolare, il "locale" dei contenuti, ha generato numerose problematiche. Questo perché era necessario effettuare delle ricerche anche solo in alcune lingue. La sua struttura si divide infatti in 2 livelli, lingua in formato ISO 636/2, poi ogni lingua possiede un nome, una descrizione e nuovamente la lingua nel medesimo formato. Il numero di lingue assegnate ad un cliente però, può cambiare nel tempo, come le lingue supportate nell'intera piattaforma. Nel caso di MongoDB, il *main repository*, avere un'oggetto contente un insieme di attributi variabile nel tempo come in questo caso non genera nessun problema, ma gli indici di Elasticsearch devono, come già sopra descritto, avere una struttura ben definita di un attributo per poterne cercare tutte le proprietà. Si sono valutate diverse soluzioni a questo problema, ma erano inefficienti, perché prevedevano array di oggetti o di perdere la correlazione tra elementi. In comune accordo con l'azienda, ho inserito quindi tutte le lingue ora disponibili su THRON nell'indice, trattandole come proprietà di "locale". Il lato negativo di questa decisione, è che quando sarà necessario aggiungere una lingua al sistema, sarà anche necessario modificare l'indice ed effettuare la reindicizzazzione di tutti i contenuti. Questa soluzione è stata poi utilizzata in tutte le proprietà locale presenti anche nelle altre collezioni.
Per i campi testuali che sono soggetti a ricerca full-text, si è scelto di utilizzare due *analyzer* differenti per la ricerca e per l'indicizzazione entrambri non key sensitive. Il perché di questa scelta, è costretto dal requisito di effettuare l'operazione di *matching* anche nelle sottostringhe delle singole parole e sono:

- Un analyzer con tokenizer di tipo **Partial Word** chiamamto **ngram** per l'indicizzazione del contenuto;

- Un analyzer con tokenizer di tipo **Word Oriented** chiamato **whitespace** per l'analisi del testo inserito come criterio di ricerca.

Per l'indicizzazione la scelta era tra i tokenizer "ngram" ed "edge_gram", ed inizialmente è stato scelto **ngram**, perché copre in modo totale i risultati, anche se potrebbe comportare delle prestazioni peggiori. Per spiegare come agiscono i *tokenizer* di questo tipo, è necessario fare un'esempio con la frase "Ciao, sono io".

Tabella 3.2: Modalità di tokenizzazione

Tokenizer	frase tokenizzata
whitespace	ciao-sono-io
ngram	c-i-a-o-ci-ao-ia-cia-iao-ciao
edge_ngram	c-ci-cia-ciao-s-so-son-sono-i-io

Come si può vedere, mentre la tipologia "whitespace" genera un array di parole, gli altri due, generano un array di stringhe, formato nel primo caso da tutte le sottostringhe presenti nel testo, mentre il secondo le sole sottostringhe che partono dalla prima parola. Siccome questo metodo di tokenizzazione comporta anche l'aumento del numero di operazioni di confronto tra stringhe durante la fase di ricerca, è possibile sezionare in entrambi i casi il numero di lettere dal quale partire e quello fino a dove arrivare. Il numero di lettere sottoposte a tokenizzazione è stato sottoposto a valutazione considerando la soddisfacibilità dei requisiti. Il minimo numero di lettere è stato impostato a 1 , mentre il massimo a 20. Questo implica che tutte le parole con un numero di caratteri maggiori a 20, daranno risultati di ricerca vuoti.
Il perché dell'utilizzo di due *analyzer* distinti diviene ora ovvio. Per soddisfare i requisiti, bisogna ricerca le singole parole del testo o l'intera frase, all'interno dei token definiti nell'indice. Se utilizzassi lo stesso tokenizer le parole in ingresso durante la ricerca, avverrebbero dei match anche su frazioni di parole non significative o differenti.

3.3.3 Sviluppo del servizio

Prima di iniziare la progettazione e lo sviluppo del servizio riguardate la POC con Elasticsearch, ho effettuato, in accordo con il mio tutor aziendale dello *scouting* sui servizi dei maggiori servizi presenti nel Web. Grazie a quest'operazione ho redatto un piccolo *report* riguardante come altre aziende, di concorrenza o meno offrivano i loro servizi di ricerca. Alcune soluzioni erano semplici e intuitive, altre meno, ma offrivano delle enormi potenzialità.
Il risultato è stato un accorgimento sia a livello di gergo che implementativo. In tutti i servizi di ricerca full text, si fa una netta differenza tra ricerca testuale e filtri del risultato. La prima difatti è una sola in un'operazione di ricerca, e ciò è anche necessario per poter fornire gli highlight previsti nei requisiti, mentre i filtri sono delle restrizioni alla ricerca basate su altre proprietà nel quale non è svolto quel tipo di ricerca.
Dopo aver affrontato questa ricerca, ed effettuato i test sugli indici di Elasticsearch, ho iniziato ad affrontare la progettazione della lambda per il POC. La prima scelta è stata su quali tecnologie utilizzare. Come precedentemente descritto la scelta è ricaduta su **Python 2.7** come linguaggio di programmazione, questo per la disponibilità di un framework chiamato **Chalice**. Quest'ultimo è stato valutato e scelto, su consiglio di un collega, perchè consente di creare e distribuire rapidamente applicazioni che utilizzano Amazon API Gateway e AWS Lambda. Fornisce uno strumento di riga di comando per creare, distribuire e gestire l'applicazione, un API familiare e facile da usare per la dichiarazione delle viste nel codice in produzione ed una generazione automatica dei criteri AWS IAM.
Per l'interrogazione del database, ho deciso di usare direttamente l'interfaccia REST messa a disposizione da Elasticsearch, senza utilizzare librerie di mediazioni. Questo permette di costruire manualmente le query, e avere quindi il pieno controllo della richiesta, cosa molto importante per un POC che vuole valutare le capacità di uno strumento.

In seguito ho definito l'interfaccia della API in formato OpenAPI mediante l'uso di **Swagger**. Durante la definizione delle api, ho stabilito due chiamate HTTP per la ricerca per ogni entità:

- **GET** per l'auto completamento, offrendo un numero limitato di opzioni, e considerandola quindi una ricerca più semplice;
- **POST** per la ricerca avanzata, permette difatti di inserire un maggior numero di opzioni per il filtraggio dei risultati.

In questo momento, ho deciso anche, quali fossero i tipi di ricerca full-text che, sarebbero stati supportati per soddisfare i requisiti e funzionalità richieste. I tipi di ricerca scelti, sono stati 3:

- **AND**: Nel risultato della ricerca, tutti i token generati dall'analisi di Elasticsearch devono essere presenti;
- **OR**: Nel risultato della ricerca, basta che sia presente un solo token generato dall'analisi, ma maggiore è il numero di token, presenti, maggiore darà lo "_score" del ranking;
- **PHRASE**: Nel risultato della ricerca deve essere presente l'intera frase, come se fosse un unico token.

La prima coppia di API sviluppata, ha riguardato la ricerca sui contenuti. Ho proceduto quindi alla definizione delle richieste e delle risposte, prevedendo un attributo con la ricerca testuale e altri attributi per i vari filtri consentiti e previsti dalle richieste aziendali. Questo tipo di richiesta permette di avere anche una gestione degli *highligth* semplificata, e di generare la query per il database senza iterpretazioni forzate.
Ho deciso inizialmente di far supportare al servizio anche la ricerca fuzzy, per avere un paragone di esecuzione, con la normale ricerca, inserendolo quindi come parametro numerico all'interno dell'API, questo rappresenta il numero di lettere errate che possono essere contenute nella frase. Per quanto riguarda l'organizzazione dei risultati invece, è stata prevista una semplice paginazione di tipo "offset-limit" per quel numero di elementi, utilizzabile attraverso "query params" in entrambe le chiamate. Con questo tipo di paginazione si aumenta la testabilità, e si ha un raffronto nella modalità meno efficiente. In seguito a proceduto a sviluppare le api per le altre entità previste, nelle stesse modalità, portando incrementi consecutivi al prodotto.

3.3.4 Test di ricerca e performance

Ho suddiviso i test di validazione effettuati in due tipologie:

- **Test di funzionamento**: effettuati manualmente, con una serie di input scelti alle api, con il software Postman, che permette il salvataggio delle chiamte http e con la funzionalità test, di automatizzare il controllo delle risposte.

- **Test di performance**: effettuati nel medesimo modo, ma dando per scontato il corretto funzionamento, e concentrandosi sul tempo di risposta della chiamata.

I test di funzionamento, sono serviti in particolar modo durante la fase di sviluppo del servizio, e sono stati effettuati per la maggior parte quando il servizio era in esecuzione localmente, e su tutte le entità. I test di performance invece, sono stati effettuati solo ed esclusimante quando il servizio era in esecuzione su AWS Lambda, ed interessando principalmente i contenuti. Questa collezione gode maggior interesse nell'ambito per la quantità rilevante di documenti che contiene.
Inizialmente, nell'indice dei contenuti, presente in elasticsearch, erano inseriti esclusivamente i dati provenienti dall'ambiente di "quality" quindi nettamente inferiori a quelli di produzione (circa 8000 elementi). In questo caso, il carico per il servizio di Elasticsearch durante la ricerca era pressochè nullo e per questo motivo sembrava funzionare tutto perfettamente, con gli indici precedentemente sviluppati.
Sfortunatamente una volta importati i dati dall'ambiente di "produzione" (più di 5 milioni di elementi), per i test di performance, c'è stato un netto calo delle prestazioni. Indagando la motivazione del calo di prestazioni, ed effettuando le chiamate di test dalla rete Aziendale, ho deciso di iniziare ad utilizzare lo strumento fornito da Amazon per il *monitoring* delle Lambda. Grazie a questo ho potuto notare, che era proprio il tempo di esecuzione della Lambda a protrarsi anche per qualche decina di secondo. Questa latenza, rendeva lo strumento e il POC da me sviluppati invalidi per l'obbiettivo aziendale.
Successivamente ho provando ad eseguire la medesima query direttamente ad Elasticsearch, mediante lo strumento **Postman**, per verificare se il problema fosse effettivamente legato all'operazione di ricerca. Ho provato a verificare se la query fosse troppo complessa, ma anche con una ricerca di un testo semplice su un attributo semplice, la latenza rimaneva pressochè invariata. Dopo aver eseguito il test, sono giunto alla conclusione che fosse un problema di indicizzazione, e che quindi lo schema dell'indice *contents* andasse modificato.
Il problema quindi poteva risiedere molto probabilmente, nel tipo di tokenizzazzione. All'aumentare del numero di documenti difatti, il numero di token generati dalla tokenizzazzione di tipo "ngram" risulta crescere in modo repetino. Ho provato quindi a reindicizzare i contenuti utilizzando la modalità "edge_ngram". Dopo un raffronto matematico, ho potuto sviluppare 3 semplici formule che mostranto l'aumentare di token in base al numero di parole e lettere medie per parola. Questo dato è solo indicativo in quanto non ben definito e mostra l'aumento del carico per il database a seconda delle due tokenizzazioni.

`a = numero di lettere medie per parola`

`b = numero di parole`

$$\texttt{ngram - b} * \sum_{k=1}^{a} k$$

```
edge_ngram - b * a
```

Rieseguendo i test, questa volta, le prestazioni erano più incoraggianti. I tempi erano scesi da più di 10 secondi a quasi 1 secondo. Per rendere più appetibile questo risultato, ho poi ragionato sul numero di lettere minime e massime con il quale effettuare la tokenizzazzione. Basandomi sul fatto che nella maggior parte delle lingue, le parole di un sol carattere hanno uno scarso significato semantico, ho pensato di escluderle dalla ricerca. Questo ha portato un notevole aumento di prestazioni. Per fare ciò però ho dovuto richiedere una modifica dei requisiti al mio tutor, che ha convocato una riunione prima dell'appovazione. Dopo aver ricevuto il via libera, ho provveduto a modificare il servizio per rimuovere le porzioni di testo che altrimenti avrebbero fatto restituire un risultato vuoto alla *query* Elasticsearch.
In seguito ho ripetuto entrambe le tipologie di test, per verificare il miglioramento. nella tabella in seguito, presento i risultati medi dei test di performance eseguiti nei 3 momenti sull'indice "Contents" contenente i dati di produzione. per ogni uno dei client id, è stata effettuata una misurazione su ricerca di tipo AND , OR, PHRASE, con e senza fuzzy.

Tabella 3.3: Tempi risposta cache distattivata

Caso	**N. lettere da - a**	**Risposta peggiore (ms)**	**Risposta media (ms)**	**Risposta migliore (ms)**	**Spazio indice (GB)**
ngram	**1 - 20**	21.396	12.213	8.232	9
ngram	**1 - 9**	7.097	4.532	2.295	6
edge_ngram	**1 - 20**	1.392	911	730	4.87
edge_ngram	**2 - 20**	916	617	412	4.17
servizio attuale	//	5.781	1.252	1.118	//

Per rendere il risultato più simile ad un caso reale, i test si sono svolti su 6 diversi clienti, quelli maggiormente ricchi di informazioni. Il cluster utilizzato presso il servizio di Amazon ha avuta una taglia minima, per evitare che il dato venisse falsato una volta sottoposto a *query* da differenti persone. Le informazioni precedentemente riportate e quelle in seguito, non presentano quindi quelle utilizzate per il dimensionamento del servizio.
Terrei a precisare inoltre, che la maggior parte dei risultati peggiori, si verifica quando si utilizza a ricerca fuzzy, questo perché Elasticsearch deve analizzare più token rispetto alle altre tipologie. Quella di tipo **phrase** risulta essere invece la più rapida.
Dai test eseguiti, si è inoltre notato che Elasticsearch sfrutta in modo ottimale il "clientId" ovvero il filtro per cliente nel caso di cache attiva. Difatti Elasticsearch esegue prima le operazioni di filtro, e poi le operazioni full text, di conseguenza se la cache è attiva, esso

esegue anche 5 volte più velocemente una *query* su un "clientId", se ne è stata eseguita una nello stesso recentemente. Per questo motivo, prima di ripetere la sequenza di test, la *cache* veniva ripulita, per evitare di imbattermi nel caso ottimo, e basarsi solo sul peggiore. In seguito riporto un grafico di chiamate successive dello stesso cliente, che mostra come il tempo di interrogazione in un primo momento cala, per poi stabilizzarsi una volta entrato in cache.

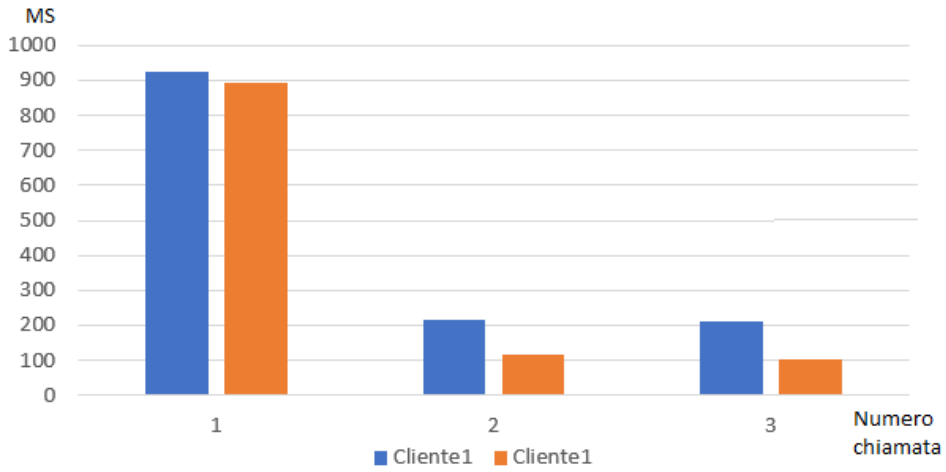


Figura 3.5: Grafico prestazioni con cache

3.4 Sincronizzazione database

3.4.1 Analisi di requisiti

Con il POC precedentemente effettuato, è sorta l'esigenza, già prevista in passato, di mantenere aggiornato Elasticsearch con i documenti contenuti nel *main repository* MongoDB. E' stato quindi possibile definire con più precisione, le esigenze che la sincronizzazzione dovrà soddisfare.
Uno dei requisiti che risulta subito rilevate riguarda il formato del documento. Difatti il documento non è uguale nel formato all'interno dei due database. E' quindi necessaria una conversione prima dell'importazione di quest'ultimo.
Altro requisito importante è che permetta di iniziare la sincronizzazione da un *timestamp* specificato, dopo avere effettuata una massiva precedentemente. Durante i test di indicizzazione, si è anche notato che l'operazione di importazione dei dati può durare anche moltissimo tempo, data la grande quantità di dati da convertire ed inserire.
Le richieste aziendali prevedono una sicronizzazzione che può avvenire anche in tempi distinti in base alle entità. La cosa più importante nell'aggiornamento sono le modifiche ai permessi di visualizzazione (le ACL). Questo è necessario per non generare confusione all'utente, offrendo un contenuto che poi non potrà visualizzare.

I requisiti di alto livello precedentemente trovati, verranno riscritti nella forma tabellare a seguito. Per la loro classificazione, non ho utilizzato una nomenclatura particolare, ma li ho semplicemente numerato in ordine crescente.

Tabella 3.4: Requisiti di sincronizzazione

Identificativo	Descrizione
1	Modifica del documento durante la fase di importazione
2	Partenza della sincronizzazione da un *timestamp* definito all'avvio
3	Deve essere possibile bloccare e di avviare la sincronizzazione
4	La sincronizzazione deve essere indipendente dal cliente
5	Deve essere possibile scegliere quali database e collezioni effettuare la sincronizzazione

3.4.2 Sviluppo e soluzioni adottate

Andando a verificare i requisiti, il primo problema che mi sono posto era se esistesse già qualche soluzione in commercio che offriva tali servizi. Ho trovato alcuni software, molte delle quali però erano a pagamento, non offrivano un periodo di prova gratuito, non ho potuto quindi verificare che risolvessero la mia problematica, anche se quasi tutti di questi parlano di due copie "identiche" dei database.
Successivamente ho trovato un progetto su GitHub di MongoLab, la società che mantiene il progetto di MongoDB. Quest'ultimo consiste in un software Python chiamato "mongo connector" che leggendo la collezione degli **Oplog** di MongoDB, invia l'evento rilevato ad un'interfaccia, per la sincronizzazione con altri database, uno dei quali era Elasticsearch. L'**Oplog** è una **capped collection**, ovvero una collezione con una dimensione prestabilita in termini temporali e numerici, che viene utilizzata da MongoDB per la sincronizzazzione dei suoi nodi "slave". In questa collezione vengono salvati dei documenti contenenti le operazioni di inserimento, modifica e cancellazione di documenti, collezioni ed indici. Con questo software è quindi possibile collegarsi ad uno dei nodi slave per poi leggerne l'oplog.

Mongo connector è diviso in due porzioni, la prima, che si occupa di una sequenziale lettura dell'Oplog, mentre la seconda chiamata **"doc_manager"** ha il compito di apportare le modifiche all'altro database.

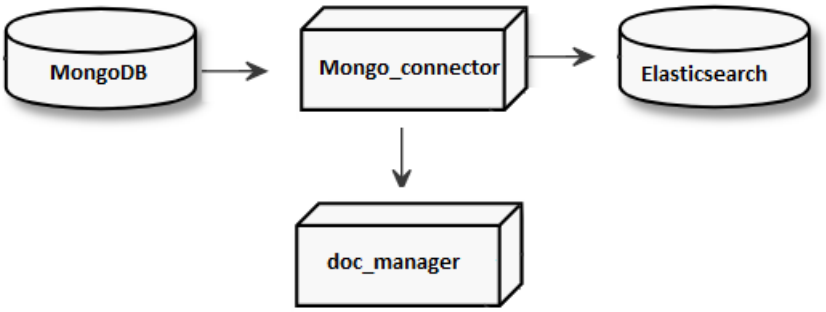


Figura 3.6: Architettura di Mongo_connector

In accordo con il tutor aziendale e la figura di supporto a me assegnata, ai quali ho fornito pregi e difetti della soluzione, abbiamo deciso di utilizzare questo software, per l'allineamento dei due database.
Sfortunatamente però, non era disponibile un doc_manager per AWS Elasticsearch Service nella versione 5. Essendo però il progetto open source, con un'interfaccia ben documentata, in comune accordo con il tutor aziendale, abbiamo deciso di creare un doc_manager appropriato alle esigenze del servizio.
Solo alcune funzioni previste nell'interfaccia del doc_manager sono utili alla casistica. Le funzioni dell'interfaccia che ho implementato sono riportate nella seguente tabella, con relativa descrizione.

Tabella 3.5: Funzioni implementate nel document manager

Funzione	Descrizione
Update	permette di ricevere le modifiche del documento e l'id dello stesso
Upsert	permette di ricevere il documento caricato e l'id con il quale è stato caricato
Remove	permette di ricevere l'id del documento già rimosso

Per semplificare il processo di conversione delle modifiche MongoDB, che nella procedura di update mostrava solo la query di modifica e non il documento dopo la sua modifica, ed avere un dato sempre aggiornato, ho reputato come soluzione migliore, chiedendo consulto anche al mio referente aziendale, di reperire il dato attraverso l'**_id** di MongoDB, che è univoco. In questo modo si ha sempre l'ultima versione del documento da inserire all'interno del database Elasticsearch. Le problematiche dovute a problemi di sincronia, rimozione e inserimenti dopo rimozioni, sono risolti grazie a questo accorgimento, e con un'accurata gestione degli errori, vengono gestiti anche casi nel quale l'ordine delle operazioni non è quello avvenuto in precedenza. Facendo un'esempio:

```
inserimento + update + rimozione
```

- Nel caso in cui la rimozione venga registrata prima dell'inserimento, l'operazione non avverrebbe, perchè il documento non esisterebbe nel database di Elasticsearch.
- Viceversa se inserimento e update vengono già registrati dopo che è stata effettuata l'eliminazione nel *main repository*, il documento non verrà mai inserito o aggiornato in Elasticsearch.

Lo sviluppo del prodotto è poi stato suddiviso in due fasi. La prima fase ha previsto lo sviluppo di un doc_manager che permettesse l'inserimento delle modifiche e dei documenti in Elasticsearch5, e la produzione di uno script che permettesse l'avvio del connettore da un punto preciso del tempo (timestamp). L'intero prodotto è stato infine posto all'interno di *Docker*, offrendo tramite variabili d'ambiente una serie di impostazioni utili come il timestamp da cui partire. La seconda invece aveva come scopo la separazione del document manager in 2 software distinti con compiti differenti, comunicanti attraverso il servizio **SQS** di Amazon.

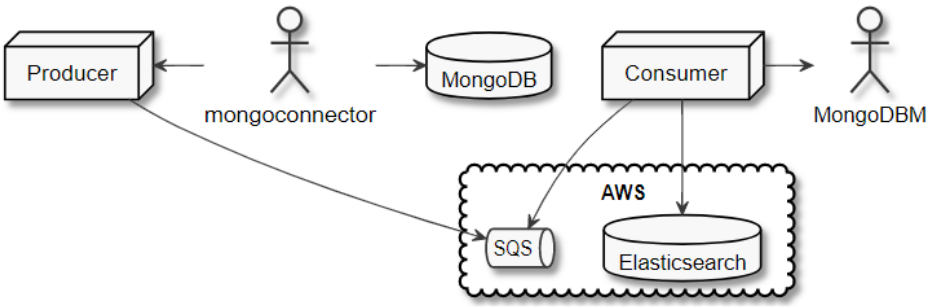


Figura 3.7: Architettura di sincronizzazione con SQS

Il document manager avrà ora il compito di inviare il Log(modifica) in un formato JSON concordato in *Simple Queue Service* di AWS (Amazon SQS), contente l'identificativo del documento e il tipo di log.
Il secondo software avrà invece il compito di prelevare la modifica da Amazon SQS, prelevare il documento aggiornato dal Main repository, attraverso un nodo slave, modificarne il formato e inserirlo o aggiornarlo in Elasticsearch Service.

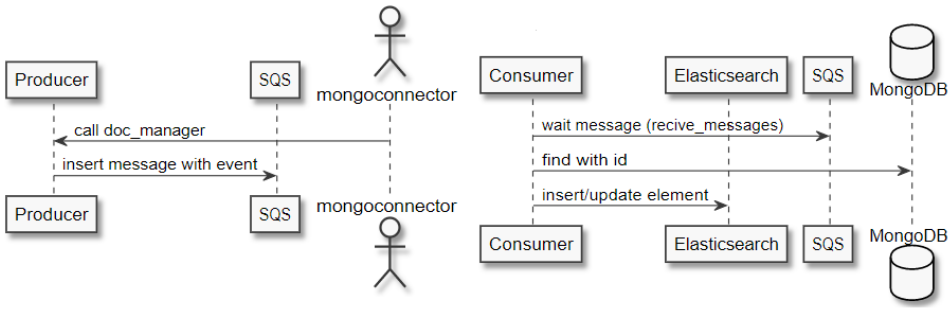


Figura 3.8: Sequenza di sincronizzazione con SQS

Ho fatto questa scelta architetturale per ammortizzare i carichi improvvisi all'Oplog. L'oplog come prima spiegato, ha un limite di documenti e tempo per il quale le modifiche permangono. Consultandomi con i colleghi dell'azienda, mi è stato riferito che in alcuni periodi di tempo, avvengono delle massive scritture nel database che riducono la permanenza delle modifiche dell'oplog a tempi molto ristretti. L'avere una coda scalabile in performance e spazio come SQS che funge da buffer, permette non solo la riduzione delle problematiche legati alla produzione di una soluzione ad Hoc, ma anche la scalabilità in termini di numero dei software di lettura ed inserimento, nel caso fosse necessario. Per necessità di riduzione dei costi e testing, è stata anche inserita la possibilità di ridurre tramite un'apposita variabile d'ambiente nel document manager il numero di collezioni sottoposte a sincronizzazione.
Il collaudo e il testing dei due software è stato fatto con un'apposita collezione e database nell'ambiente di sviluppo. Era necessario difatti necessario assicurarsi che le modifiche apportate in MongoDB, venissero effettuate anche in Elasticsearch senza l'utilizzo di stub. Per fare ciò sono state predisposte delle query di inserimento, modifica e cancellazione per MongoDb e poi di lettura per Elasticsearch dopo un certo periodo di tempo.

3.5 Sviluppo DAO per microservizio di ricerca

Terminato lo sviluppo del software di sincronizzazione, mancavano ancora due settimane al termine dello stage.
Le scelte possibili erano due: la prima permetteva di migliorare il POC, rendendolo più simile al possibile servizio reale, oppure iniziare la produzione di una porzione del futuro servizio. La scelta da parte del mio tutor è stata quella di incaricarmi dello sviluppo della porzione DAO del futuro servizio, ma prima abbiamo effettuato alcuni dialoghi, per capire come definire le api del servizio, che lui essendo anche architetto avrebbe rifinito. Valutando pregi e difetti, in base anche allo scouting effettuato in precedenza, ho consigliato di mantenere un solo attributo nella richiesta testuale, per il problema negli higlight già citato, mentre per i filtri, abbiamo optato per una struttura fissa, inseribile più

volte in 3 array:

- **must** : Filtri che devono essere veri nell'elemento ricercato;
- **should**: Filtri del quale almeno uno deve essere vero nell'elemento ricercato;
- **must_not**: Filtri che non devono essere veri nell'elemento cercato.

Quest'attributo prevedeva tutti i campi "filtro", ma solo uno di questi poteva essere presente contemporaneamente.
Questa base del microservizio, ha come requisito fondamentale quello di essere sviluppato nel linguaggio Scala, attualmente in uso nella parte core di THRON. Prima di iniziare lo sviluppo e la progettazione, ho dovuto studiare le basi di scala e gli strumenti più appropriati per lo sviluppo, nonchè le norme e le librerie comunemente utilizzate in azienda. Durante questo periodo ho appreso dal mio referente aziendale, la presenza di un microframework, che permette la strutturazione di un servizio Scala, attraverso l'auto-generazione dei file necessari. Consultando proprio quel progetto, è stato deciso il suo utilizzo in accordo con il tutor aziendale e il referente.
Durante la progettazione, ho valutato differenti framework per l'interrogazione di Elasticsearch, ma l'opzione migliore è sembrata comunque essere l'utilizzo di semplici chiamate http verso l'interfaccia REST di Elasticsearch per il suo utilizzo. La motivazione è la stessa del POC, le librerie a disposizione per interrogare Elasticsearch non permettono una sufficiente personalizzazione delle query, e riducono quindi le performance e l'espressività. Per effettuare le chiamate http, ho deciso di usare AKKA, perché quella già inclusa nel progetto che faceva parte delle migliori librerie che avevo reperito. Per la conversione da CASE CLASS a JSON, ho utilizzato invece una libreria chiamata **Circe**. Essa permette una rapida conversione delle case class, ma anche di impostare delle conversioni implicite che verranno utilizzate durante il processo.
Per rendere però il DAO semplice da utilizzare, mi è stato chiesto di produrlo con una semplice **grammatica d'interrogazione**. Studiando nel linguaggio scala, è disponibile ai programmatori uno strumento cruciale per lo scopo, le conversioni implicite. Grazie a queste ho potuto definire delle conversioni da un tipo tupla a due o più elementi ad un oggetto utilizzato per la ricerca. Ho definito quindi una serie di case class (classi distanziabili, serializzabili e formato da solo attributi) rappresentati i vari tipi di operazione (search operator) effettuabili per ogni tipo di dato. Per riuscire a controllare a tempo di compilazione se un search operator venisse utilizzata in un capo compatibile, ho generato anche una serie di interfacce (trait) rappresentanti i vari campi dell' indice sottoposto a ricerca. Grazie a Circe ed alle relative conversioni implicite, mi è stato possibile trasformare gli oggetti di richiesta, in un'oggetto JSON, che consisteva nella query di interrogazione per Elasticsearch.
Nel codificare questa parte di servizio, ci ho messo molto più di quanto previsto, a causa principalmente della mia scarsa conoscenza di scala, soprattutto nel meccanismo delle conversioni. Durante lo sviluppo, per accertarmi della funzionalità del servizio, ho codificato dei test automatici, utilizzando "ScalaTest". Il mio obbiettivo è stato di offrire

una copertura quasi totale nel numero di *branch* (>85%), proprio per far fronte alla mia scarsa conoscenza di Scala, e di alcune sue peculiarità. In questo modo ho confermato che il servizio eseguisse secondo le aspettative e anche che il tempo di esecuzione, si presentasse simile a quello del microservizio lambda.
L'ultima attività da me svolta, è stata la scrittura in swagger dell'interfaccia API del futuro servizio di ricerca una volta rifinita dal mio tutor. Quest'attività, sarebbe servita in futuro, per la generazione della parte mancante del servizio, e la sua documentazione. Quest'ultima attività, della durata di un giorno, si è chiusa con l'approvazione dello schema da parte del mio referente interno.

Capitolo 4

Conclusioni

4.1 Soddisfacimento degli obiettivi

Come illustrato precedentemente nella sezione 3.1, prima dell'inizio dello stage, ho redatto un piano di lavoro che descriveva le attività che avrei svolto. Ho svolto quest'operazione in collaborazione con il tutor aziendale.
Gli obiettivi si concentravano sulla produzione del POC (*Proof of concept*) che avrei sviluppato per il servizio di ricerca con Elasticsearch e il software che permette la sincronizzazione tra MongoDB e il database utilizzato durante la ricerca. Il software di sincronizzazione è reso necessario per poter mantenere allineati i due database durante la ricerca. Elasticsearch o altri *search engine* hanno un proprio database per poter eseguire la ricerca, questo ne rende quindi necessario la sincronizzazione con il main repository. Riporto in seguito una tabella che rappresenta gli obiettivi fissati prima dello stage ed il loro stato.

Tabella 4.1: Soddisfacimento obiettivi stage

Identificativo	Descrizione	Stato
O01	Definizione DB e schema per registrazioni metadati contenuti.	Soddisfatto
O02	Definizione struttura servizi per le ricerche (query language, parametri ricerca, struttura oggetti di risposta del servizio)	Soddisfatto
O03	AWS Elastic Search: importazione dati da Database Mongodb. Definizione struttura collezioni ed indici, performace test su dati destrutturati e su dataset da 10 milioni di documenti.	Soddisfatto
O04	Sistema di sincronizzazione tra main repository (Mongodb) dei contenuti ed AWS Elastic Search.	Soddisfatto
O05	Definizione test search webservice in AWS: API Gateway + Lamba Function+ ElasticSearch (linguaggio Scala).	Soddisfatto

Alla fine dello stage sono riuscito a soddisfare completamente gli obiettivi riguardanti il software di sincronizzazione con tutti i suoi requisiti. La persona a me affiancata e il tutor, hanno espresso pareri positivi sul prodotto sviluppato. Quest'ultimo è pronto per un futuro test in ambiente "quality" prima di un'eventuale utilizzo. Questo periodo di testing va effettuato per la mancanza di test di carico accurati.
Per il POC del servizio di ricerca invece, non ho potuto fare un raffronto completo tra quelle che saranno le prestazioni finali e quelle che sono quelle dell'attuale servizio. La mancanza di questo raffronto è dovuta alla sincronizzazione tra il database di test e quello utilizzato per la ricerca. Al tempo di risposta della lambda riguardante il POC difatti andrebbero aggiunte due operazioni: *getThumbnail* e la ricerca mediante "_id" del documento in MongoDB, che comportano un piccolo *overhead* nella restituzione del risultato. Questa lacuna è dovuta all'impossibilità di inserire il software di sincronizzazione appena creato nell'ambiente corretto nel quale reperire i dati.
Dal punto di vista funzionale il POC è completo e permette la ricerca secondo le modalità definite durante lo svolgimento del progetto. Un'operazione che sfortunatamente è impossibile, per motivi di prestazioni è la ricerca su tutti i grammi dei testi. Come spiegato precedentemente, la ricerca di tipo *ngram* sarebbe poco scalabile sul numero di elementi. Quello della scalabilità sulla quantità di elementi è fondamentale data la previsione di un aumento di contenuti nel breve futuro.

Con il tempo rimasto dopo il test del POC, circa 7 giorni, in accordo con il tutor, ho ritenuto opportuno procedere alla produzione della parte DAO con Elasticsearch del futuro servizio di ricerca. Ho sviluppato il servizio del POC seguendo le interne convenzioni aziendali e cercando di soddisfare tutti i requisiti richiesti nel POC o quelli aggiunti successivamente. Questo sfortunatamente non è stato sottoposto ancora a valutazione aziendale, in quanto è stato prodotto a ridosso della scadenza dello stage.
Nel complesso il tutor aziendale e le figure con il quale ho comunicato all'interno dell'azienda, sono state soddisfatte della sperimentazione effettuata. Da questa ho potuto ricavare dati utili riguardo le tecnologie sperimentate, le possibilità di ricoprire i servizi richiesti dal cliente e le modalità con il quale tenere aggiornato MongoDB con altri servizi.

4.2 Bilancio formativo

L'esperienza di stage è stata un'ottima opportunità di confermare le competenze acquisite durante il mio corso di studi e di apprendere cose nuove. Oltre all'acquisizione di competenze tecniche, mi ha permesso di imparare cosa significhi lavorare a contatto con altre persone in un contesto aziendale.
Riguardo le competenze tecniche acquisite, durante lo stage ho potuto apprendere nuovi linguaggi di programmazione come Python 2.7. Ho avuto modo anche di imparare molto riguardo la programmazione funzionale in Scala e delle diverse modalità di utilizzo di questo. Esistono difatti tecniche per sfruttare al meglio i vantaggi offerti dai linguaggi funzionali, che precedentemente non avevo avuto occasione di conoscere. Alcuni esempi di questi vantaggi possono essere:

- la possibilità di utilizzare come argomento le funzioni stesse che permette il meccanismo delle promise/callback, agevolando programmazione concorrente;
- la possibilità di differenziare tra oggetti immutabili e non, che permette una maggiore sicurezza durante la programmazione;
- la presenza di alcune strutture dati come le tuple, che semplificano il numero di parametri delle funzioni oltre che permettere di ridurre il numero di variabili.

In particolare, sono rimasto colpito dalle opportunità che offre Amazon AWS. Lo avevo già utilizzato in precedenza durante il progetto didattico di Ingegneria del software e per l'hosting di alcuni servizi Javascript. Però non avendo mai visto un progetto di dimensioni paragonabili a THRON, non avevo mai capito appieno le potenzialità di AWS soprattutto nell'ambito service. Ora dopo aver visto i numerosi vantaggi che derivano dall'utilizzo di servizi come **SQS**, ed **Elasticsearch as a service**, ne comprendo molto di più l'importanza. Le caratteristiche che hanno attirato più la mia attenzione sono state la scalabilità e le basse necessità di manutenzione da parte dell'utilizzatore, mansioni del quali si occupa Amazon. Considerando i progetti di piccole dimensioni, non avevo mai prestato molta attenzione alla scalabilità e quindi non avevo mai potuto apprezzare

questa caratteristica.
Considero di particolare importanza anche l'acquisizione di conoscenza sull'elaborare la ricerca in grandi quantità di dati. Non avendo mai lavorato con quantità così ingenti di dati, non avevo mai compreso quanto potessero degradare le prestazioni di un servizio come Elasticsearch, all'aumentare dei contenuti. Per gli stessi motivi, ovvero la quantità di dati, lo stesso MongoDB durante le ricerche è molto più lento rispetto quanto avessi immaginato. Proprio per questo motivo durante questo stage, ho imparato a prevedere sempre la scalabilità del software. Considerare il caso d'uso prima della struttura del software, è molto importante. In questi deve essere considerata anche la quantità di dati. Dopo quest'esperienza so che non sempre la struttura che sembra essere la più semplice ed efficace o completa si rileverà essere la migliore per quel caso d'uso. Quest'ultimo potrebbe comportare anche una riduzione dei requisiti, o aggiungerne altri che precedentemente non erano previsti.
Lo stage è stata anche un'ottima esperienza per poter valutare una delle occupazioni più comuni per i laureati del mio corso, quella del programmatore. Questo mi ha aiutato a far chiarezza su quale fosse il percorso che intendevo affrontare dopo lo stage e la seguente laurea.

4.3 Distanze tra mondo universitario e lavorativo

L'università ha il difficile compito di formare persone che, una volta uscite dall'università, possano entrare facilmente nel mondo del lavoro. Questo compito è reso difficile dall'ampia gamma di strade diverse che l'indirizzo di informatica offre e dalla continua evoluzione del settore. Infatti l'informatica, è una disciplina in continuo e rapido sviluppo, che rende difficile trovare i giusti compromessi sulle nozioni da impartire agli studenti.
Il tipo di preparazione offerto dal corso di informatica da me affrontato, è fornire una base di conoscenze teoriche, che poi possano essere utilizzate in molti contesti, ambiti e con strumenti anche molto differenti tra loro. Le nozioni base al quale mi riferisco sono fornite principalmente dai corsi di Programmazione ad oggetti, Programmazione concorrente, Tecnologie web ed Ingegneria del software.
Questi corsi, mi sono stati utili soprattutto per la presenza di progetti ad esse correlati. Infatti, se durante il corso, è coltivata solo la parte teorica della disciplina, i progetti ne completano l'apprendimento con un esercizio pratico. In particolar modo, il progetto di Ingegneria del software si è rivelato molto utile per la quantità tecnologie e discipline applicate. È proprio durante il progetto di Ingegneria del software che ho potuto utilizzare per la prima volta Amazon AWS e sperimentare la produzione di un software dall'analisi dei requisiti al suo collaudo.
Cosa che invece non trovo coerente, è il far conoscere alcuni strumenti indispensabili agli sviluppatori, come i strumenti di versionamento solo al terzo anno. Questi difatti sarebbero utili anche a progetti svolti negli anni precedenti, e meriterebbero forse di essere accennati anche da altri insegnamenti non solo da ingegneria del software. Personal-

mente ne avevo già una precedente conoscenza per motivi lavorativi, ma ne ho appreso il corretto utilizzo solo durante il progetto di Ingegneria del software.
Personalmente ho trovato sufficienti nello svolgimento del mio stage le nozioni apprese durante l'università. Ho però riscontrato difficoltà con l'apprendimento delle basi funzionali del linguaggio Scala. Per tale motivo avrei trovato utile una breve introduzione al paradigma funzionale. Quell'introduzione mi avrebbe concesso un notevole vantaggio durante l'attività di apprendimento. Questo tipo di nozioni, come i paradigmi più moderni senza entrare nello specifico di un linguaggio, potrebbero essere impartite con un seminario come quelli tenuti durante il corso di Ingegneria del software durante il secondo semestre.
Per concludere, mi sento soddisfatto della mia preparazione universitaria, e ritengo che la via intrapresa per insegnarla, ovvero spiegare i concetti base dell'informatica senza focalizzarsi sulle nuove tecnologie sia la strada migliore. Come detto prima, il settore dell'informatica è in continuo sviluppo, ed escono ogni giorno soluzioni e tecnologie nuove, spesso migliori delle precedenti. Cercare di riconcorrere l'ultima tecnologia sarebbe controproducente per la formazione di uno studente, rispetto al creare una buona base, che lo studente possa ampliare di sua volontà.

Glossario

Acl	Lista di controllo degli accessi (in lingua inglese di access control list, abbreviato in ACL) è un meccanismo usato per esprimere regole complesse che determinano l'accesso o meno ad alcune risorse di un sistema informatico da parte dei suoi utenti., 1
API	Api è acronimo di *application programming interface* ed indica un'insieme di procedure disponibili al programmatore mediante un'apposita interfaccia, per l'esplemento di un determinato compito all'interno di un programma. Nel caso specifico si intendono esclusivamente API disponibili attraverso protocollo HTTP., 1
API gataway	Servizio Amazon offerto attraverso Amazon web service, che permette la ricerzione, validazione, documentazione e inoltro ad altri servizi di chiamate HTTP., 1
asset	Qualsiasi bene di proprietà di un'azienda (macchinari, merci, ecc.), nello specifico un bene informatico come immagini, video e documenti, che possa essere monetizzato., 1
COO	Acronimo abbreviato di *Chief opertating officer*, ed è il responsbile esecutivo per il reparto esecutivo di un'azienda., 1

deploy — Rilascio al cliente, con relativa installazione e messa in funzione o esercizio, di una applicazione o di un sistema software tipicamente all'interno di un sistema informatico aziendale., 1

IDE — Ambiente di sviluppo integrato (in lingua inglese integrated development environment ovvero IDE, anche integrated design environment o integrated debugging environment, rispettivamente ambiente integrato di progettazione e ambiente integrato di debugging) è un software che, in fase di programmazione, aiuta i programmatori nello sviluppo del codice sorgente di un programma. Spesso l'IDE aiuta lo sviluppatore segnalando errori di sintassi del codice direttamente in fase di scrittura, oltre a tutta una serie di strumenti e funzionalità di supporto alla fase di sviluppo e debugging., 1

Ranking — Principio di valorizzazzione di un elemento sottoposto a ricerca, che ne conferisce un grado di correllazione con il contenuto ricercato., 1