
Università degli Studi di Padova

Facoltà di ...


*Corso di Laurea in ...*

# Thesis title
## (subtitle)


*Laureando*

**nome cognome**

*Relatore*

**Prof. nome cognome**

*Co-relatore*

**nome cognome**



Anno Accademico 20nn/20nn

A ...

Quote
*Author*

# Indice

**Sommario**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras mattis tincidunt ligula. Duis ante neque, convallis vel vulputate vel, dignissim vel enim. Proin et iaculis libero. Aliquam erat volutpat. Cras ac purus non ante ultricies scelerisque. Donec lobortis lorem imperdiet leo consequat nec iaculis velit adipiscing. Curabitur nec gravida neque. Nunc vel dui vitae ante dapibus sagittis ac non libero. Suspendisse gravida commodo arcu bibendum luctus. Nam placerat pharetra massa, aliquam rutrum arcu fermentum nec. In non ultrices ante. Pellentesque pretium, felis ac mattis condimentum, dui massa ultricies nisl, hendrerit malesuada magna risus eget dolor. Pellentesque lobortis eleifend nibh, sed gravida sem fringilla eget. Proin pretium, arcu in ornare pellentesque, elit ante faucibus sem, at convallis eros ante ut velit. Donec ornare erat non diam tristique vitae congue nulla commodo. Proin fermentum fringilla mattis. Pellentesque ut dolor hendrerit tellus tincidunt egestas at sit amet velit.

# L'azienda

## Presentazione

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

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

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

1. Engineering
2. Ricerca
3. Formazione

4. Product Specialist

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

1. Core
2. User Experience
3. DevOps
4. Intelligence
5. Marketplace
6. ItOps
7. Design

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

# Thron

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

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

- Video e audio on-demand
- Video e audio live

e molti altri.
Tutti questi contenuti sono raggruppati in un'unica piattaforma che analizza il contenuto e lo arricchisce di informazioni automaticamente con $\text{tag}_G$ e $\text{metadati}_G$. Questo lavoro è svolto da un motore semantico e analitico che aggiunge queste informazioni sia appena caricato il contenuto sia durante il ciclo di vita di esso. Naturalmente è possibile aggiungere, modificare o eliminare queste informazioni manualmente. Queste informazioni oltre ad arricchire i contenuti di informazioni utili aiutano anche la loro organizzazione e gestione, soprattutto quando la mole di dati inizia ad essere notevole, poichè si possono ordinare o raggruppare contenuti per tag. In TODO Figura 1 vediamo una pagina di un contenuto su piattaforma Thron.
TODO aggiungere figura contenuto con tag... Inoltre per ogni contenuto è possibile scegliere dei permessi, come scegliere che sia visualizzabile solo da alcuni utenti per renderlo privato o visibile solo da un team di interesse, si può dare il permesso di modifica del contenuto o si può garantire il permesso di condividere il contenuto in modo da delegare ad altri la condivizione all'esterno della piattaforma del contenuto.

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

1. Creazione del contenuto (chi lo ha caricato in piattaforma)
2. Modifica dei permessi e quali
3. Visualizzazioni del contenuto, utile per effettuare statistiche di chi li usa, vengono salvate anche le informazioni sul dispositivo utilizzato per la visualizzazione
4. Condivisioni del contenuto
5. Sostituzioni del contenuto, è possibile infatti caricare una nuova versione del file, la vecchia in questo caso viene automaticamente salvata nel caso si volesse tornare ad una versione precedente.

Queste informazioni sono filtrabili per diversi parametri:

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

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

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

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

Un esempio si può vedere in Figura 2. TODO aggiungere immagine history contenuto. Ogni file viene convertito in diversi formati e dimensioni in modo da ottimizzare la sua fruizione in base al canale di distribuzione finale. Ad esempio un'immagine, una volta caricata, viene copiata e ridimensionata con le varie risoluzioni che si usano maggiormanete nella piattaforma.
A seconda della banda disponibile il sistema controlla che la qualità dell'erogazione dei contenuti sia ottimale ove questo sia possibile. Ad esempio, durante la riproduzione di un video, il sistema controlla la velocità con la quale il client riceve il video e automaticamente aumenta o riduce la qualità. Naturalmente anche la qualità può essere impostata manualmente.

Il prodotto viene infine fornito a siti o servizi esterni da dei connettori, in modo da interfacciarsi e integrarsi senza modifiche alla piattaforma. Per esempio è possibile installare dei connettori per i social network più famosi o per $CMS_G$ come WordPress all'interno di Thron. Questo permette di gestire contemporaneamente un contenuto ovunque esso sia usato. Ad esempio se viene caricata una nuova versione del file questo viene automaticamente modificato ovunque sia stato condiviso. Queste opzioni sono gestite nella sezione Shareboard in Thron come vediamo in Figura 3. TODO aggiungere figura

# Processi

Thron non usa un modello di sviluppo standard per il suo prodotto, ma il tutto è molto paragonabile ad un modello agile.
Il progetto è ormai avviato e ben consolidato perciò la sua evoluzione procede attuando piccoli passi incrementali. Questi vengono fatti interagendo in maniera molto forte con i clienti che vengono spesso interpellati per suggerire anche modifiche o proporre miglioramenti.
Periodicamente vengono fatti degli incontri con clienti ai quali vengono mostrati le evoluzioni, attraverso dei prototipi, per avere dei riscontri sulla qualità e sull'efficacia della soluzione raggiunta.

Dopo questi incontri segue un incontro interno per valutare l'andamento, discutendo sul grado di soddisfacimento del cliente e sulle eventuali proposte mosse da quest'ultimo. Sempre in questi incontri interni vengono presentate anche nuove idee per sviluppi futuri.
Tutte queste nuove idee, miglioramenti, modifiche ed errori vengono scremati dal Project Management$_G$ che le gestisce attraverso un sistema di Ticketing$_G$ aziendale.
Riassumento in un diagramma il flusso delle attività è come in Figura 4.
TODO figura

## Sistema di ticketing

Thron utilizza diversi tipi di ticket per differenziare diverse procedure documentate internamente.
La classificazione dei ticket è:

1. Ticket obiettivo: indica l'obiettivo da raggiungere al termine dell'attività, comprende casi d'uso tipici, i suoi requisiti ed i vincoli

2. Ticket funzionalità: indica la funzionalità di un prodotto che si vuole ottenere con relativa motivazione

3. Ticket attività: definisce un'attività da svolgere con relativa motivazione

4. Ticket fase: definisce il passo evolutivo della funzionalità su cui si sta lavorando

5. Ticket sprint: definisce l'unione di più fasi di lavorazione

6. Ticket lavorazione: descrive il passo evolutivo dell'attività in lavorazione

7. Ticket richiesta: descrive una richieste da parte di un cliente o di un ticket sprint

8. Ticket difetto: descrive un errore riscontrato sul prodotto.

Questi ticket possono essere assegnati direttamente dal Project Management nel caso richiedano specifiche conoscenze, altrimenti sono presi in carico da persone che hanno finito le proprie mansioni attuali e rientra nella loro sfera di competenza.

## Tecnologie

Occupandosi l'azienda di molti settori dell'informatica, ed essendo divisa in diversi team, non possiede una suite di strumenti comune per lo sviluppo, perciò ogni team decide a se gli strumenti che ritengono più opportuni per gli obiettivi e le attività che gli vengono assegnati.
In comune ci sono gli strumenti di versionamento, comunicazioni e documentazione.
Per il versionamento l'azienda ha un dominio proprio di gitlab con svariate repositories per ogni progetto.
Per la comunicazione l'azienda utilizza Microsoft Outlook per la posta elettronica, questo perchè è un software che funziona su entrambi i sistemi operativi utilizzati per lo sviluppo (Windows e OS X) e poichè consente una facile gestione di eventi e riunioni su calendario. Ogni membro dell'azienda ha un proprio indirizzo email con la forma *nome.cognome@thron.com*. Per comunicazioni più brevi si utilizza un sistema di messaggistica instantanea interno alla piattaforma Thron come possiamo vedere in Figura 5 TODO figura.
Per la documentazione viene utilizzata la suite Office, in particolare Word è utilizzato per la stesura di documenti provvisori e informali, che vengono convertiti in pdf quando il documento viene approvato. Si utilizza anche Power Point per creare presentazioni sia per riunioni interne, per spiegare più efficacemente, sia negli incontri con i clienti.
L'ambiente di sviluppo più comune in azienda è Windows, visto il grande supporto per i software utilizzati e per il vasto uso anche da parte dei clienti, è presente anche OS X in particolare nei team di Design e User Experience. Viene poi utilizzato Ubuntu per i servizi server per la maggiore stabilità e semplicità in questo ambito. Sono poi presenti e accessibili a tutti delle macchine preconfigurate con sistemi operativi e versioni di browser mirate per testare il comportamento del prodotto su ambienti meno comuni.

## Clienti

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

con sempre e soltanto le funzionalità volute. Vediamo in Figura 6 la sezione Marketplace TODO figura marketplace.
L'azienda, nel caso queste soluzioni non siano sufficienti, si propone, qual'ora possibile, di creare e confezionare soluzioni personalizzate e mirate per il caso d'uso specifico del cliente.
Le molteplici e diverse richieste da parte dei clienti contribuiscono l'evoluzione della piattaforma che deve innovarsi continuamente per soddisfarli. Ognuna di queste evoluzioni conferisce a Thron una completezza sempre maggiore.
In linea generale il client tipo di THRON cerca un modo semplice di gestire una mole di contenuti più o meno grande, di distribuirli su vasta scala sempre mantenendole il controllo.
Per avere una erogazione il più possibile efficiente l'azienda ha creato una $\text{CDN}_G$ che possiamo vedere in Figura 7 TODO figura cdn.

# La proposta di stage

## Scopo dello stage

### Introduzione

Come già detto in precedenza THRON è una piattaforma che mira a facilitare tutta la gestione dei contenuti. Un punto molto importante per THRON è il poter dare informazioni sui contenuti ai propri clienti per poter migliorare le tecniche di business e di marketing.
Questo non è fatto soltanto aggiungendo tags e metadati ai contenuti ma anche tracciando molteplici informazioni riguardanti la fruizione e le visite sui contenuti stessi sui vari canali di distribuzione con l'uso di una libreria javascript proprietaria.
Le informazioni raccolte vengono poi mostrate sotto forma di grafici di semplice lettura al cliente in apposite sezioni come si vede in Figura x.
TODO figura statistiche contenuti Queste informazioni vengono raccolte attraverso una libreria scritta in javascript. Prima dello stage la libreria prevedeva la possibilità di tracciare diversi eventi per i video e un solo evento per tutti i tipi di contenuto chiamato **load**. Questo evento poteva essere usato con diversi scopi ma nella quasi totalità delle volte era usato per tracciare gli accessi e le visualizzazioni a tutti i contenuti erogati da THRON.
Questo era possibile importando nelle pagine web la libreria dei tracciamenti e aggiungendo qualche riga di codice per inviare le informazioni necessarie. Il tutto era reso automatico nel caso di importazione del player di THRON che al suo interno contiene una libreria dei tracciamenti.
Tra i vari moduli e widget$_G$ che THRON offre ai suoi clienti troviamo il Predictive Content Reccomendation (PCR). Questo è un motore raccomandativo in grado di suggerire, in tempo reale, contenuti in base al profilo degli interessi della singola persona e allo storico delle sue precedenti navigazioni. Grazie alla capacità di THRON di capire gli interessi dell'utente, l'azione del PCR è diversa da quella dei tradizionali sistemi raccomandativi che basano i loro suggerimenti su "cosa hanno fatto o visto utenti simili". THRON

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

# Obiettivi aziendali

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

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

## Obiettivi obbligatori

- Creare dei diagrammi che indichino, con granularità oraria, il numero di richieste per la singola PCR

- Creare una lista o un diagramma che indica lo sforamento del numero di richieste rispetto al modello di business attivato
- Creare una visualizzazione dello storico delle prestazioni del PCR negli ultimi 30 giorni, con intervallo selezionabile

### Obiettivi opzionali

- Aggiungere un meccanismo di personalizzazione per i widget PCR

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

## Vincoli temporali

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

## Aspettative

L'azienda aveva un forte interesse per questo progetto, questo perché il progetto è frutto di diverse richieste da parte dei clienti.
Nello specifico i clienti hanno avanzato richieste per avere informazioni riguardanti quanto lo strumento dei raccomandazione fosse effettivamente efficace, per cosa questo significhi leggere il paragrafo TODO LINK in cui si parla dell'analisi di questo problema.
Inoltre questo progetto aprirebbe altre evoluzioni specifiche per il motore di raccomandazione che verranno discusse in seguito.

## Obiettivi personali

Il mio obiettivo più grande era affrontare uno stage che mi permettesse sia di aprire sbocchi lavorativi all'interno dell'azienda sia di arricchire il più possibile il mio bagaglio personale, affrontando sfide mai viste da me prima e che fossero richieste vere e proprie di un azienda.
A StageIT Thron mi ha fatto subito una buona impressione, proponendomi questo progetto in maniera chiara e facendomi capire che mi avrebbero fatto

lavorare in un team coinvolgendomi in tutte le attività legate al progetto così da darmi l'opportunità di ottenere una buona visione del progetto e di una buona formazione con gli altri membri.
In più mi è subito interessato poiché il progetto avrebbe avuto risultati immediati e visibili poiché in caso di esito positivo l'azienda lo avrebbe utilizzato come passo evolutivo mettendo la nuova versione in produzione per i clienti.
Questo sommato alle tecnologie da me mai utilizzate o viste brevemente nel corso di studi mi ha spinto a scegliere questo progetto.

# Lo stage

## Piano di lavoro

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

- Introduzione all'azienda e alle tecnologie di sviluppo

**Seconda settimana**

- Fine studio tecnologie e partecipazione alla fase di analisi

**Terza settimana**

- Continua fase di analisi

**Quarta settimana**

- Studio e progettazione dell'interfaccia di trasferimento dei dati

**Quinta, sesta, settima ed ottava settimana**

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

## Analisi

La fase di analisi è stata molto importante e più lunga del previsto.
Uno dei motivi che mi hanno spinto a scegliere questo progetto di stage è dato dal fatto che copriva diversi ambiti, non focalizzandosi soltanto ad esempio in uno sviluppo backend o frontend, questo ha significato anche il dover interagire con diversi team aziendali effettuando molti brainstorming con componenti di vari team per collegare il tutto.
Infatti per effettuare la dashboard e presentare i dati in maniera elegante ed

intuitiva ho collaborato col team di design e UX per lo sviluppo delle grafiche e l'integrazione in Bacheca$_G$. Per la raccolta e modellazione dei dati relativi al tracciamento delle singole raccomandazioni di contenuti ho lavorato col team di Intelligence mentre per la raccolta e modellazione dei dati relativi alla quantità di raccomandazioni effettuate dalle varie applicazioni ho lavorato col team Core.
Ho potuto suddividere quindi tutte le attività in tre fasi:

- **fase 1:** collaborazione col team di intelligente per raccolta e modellazione dati utili alla creazione dei grafici qualitativi$_G$, ovvero i grafici per soddisfare le richieste dei clienti relative alla bontà delle raccomandazioni;
- **fase 2:** collaborazione col team di core per la raccolta e modellazione dati utili alla creazione dei grafici quantitativi$_G$, ovvero i grafici per soddisfare le richieste dei clienti relative alla quantità di raccomandazioni fatte ogni ora e la comunicazione in caso di sforamento di richieste;
- **fase 3:** collaborazione col team di design e di UX per lo sviluppo dei grafici e l'integrazione di quest'ultimi all'interno del software bacheca$_G$.

## Obiettivi delle attività

Gli obiettivi iniziali dello stage erano di alto livello e andavano suddivisi in micro obiettivi durante le attività di analisi. Per questo motivo dopo tutte le riunioni e brainstorming fatti coi vari team si sono trovati obiettivi più mirati per la realizzazione dell'intero progetto.
Nel capitolo a seguire TODO LINK viene spiegato in dettaglio il come si è arrivati a queste conclusioni.

### Fase 1

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

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

**Obbligatori:**

- Dovrà essere aggiunta la funzionalità di tracciare impression$_G$ nella libreria dei tracciamenti THRON

- Dovrà essere aggiunta la funzionalità di tracciare click$_G$ nella libreria dei tracciamenti THRON
- Dovrà essere aggiunto un meccanismo per tracciare eventi come click$_G$, load$_G$ e glossimpression in automatico nella libreria dei tracciamenti THRON.
- Le modifiche da apportare alla libreria dei tracciamenti dovranno mantenere una retro compatibilità totale
- Tutte le modifiche devono essere supportate nelle seguenti versioni dei browsers:
  - Internet Explorer 9 e successivi
  - Firefox 3.5 e successivi
  - Chrome tutte le versioni
  - Safari OSX 5 e successivi
  - Android 4.1 e successivi: Chrome tutte le versioni
  - IOS 4.x e successivi: Safari 4 e successivi, Chrome tutte le versioni
- Dovrà essere modificato l'enricher$_G$ per aggiungere il supporto ai nuovi eventi
- L'enricher$_G$ dovrà salvare tutti gli eventi in indici diversi raggruppati per tipo di evento
- Aggiungere al modulo dei Reports$_G$ il supporto ai nuovi eventi
- Far si che gli altri moduli di Intelligence$_G$ ignorino questi nuovi eventi per il momento

**Desiderabili:**

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

**Fase 2**

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

1. Diagramma che indica con granularità oraria il numero di richieste per la singola PCR

2. Lista o diagramma che indica lo sforamento del numero di richieste rispetto al modello di business attivato

**Obbligatori:**

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

**Desiderabili:**

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

**Fase 3**

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

- Realizzare un grafico che indichi il numero di raccomandazioni fatte da un applicazione raccomandativa master (grafico quantitativo)
  - Questo grafico deve permettere la scelta di diverse granularità, in particolare oraria, giornaliera e mensile
  - Questo grafico deve permettere la scelta di una data di inizio e di fine per filtrare i dati
  - Questo grafico deve poter mostrare, con possibilità di filtraggio, il numero di raccomandazioni delle singole applicazioni raccomandative slave

- Per ogni applicazione slave devono essere realizzati altri grafici per analizzare la bontà delle raccomandazioni (grafici qualitativi)

  - Un grafico che mostri i cinquanta contenuti raccomandati visti più volte (ovvero con più impression$_G$) con la relativa quantità di click sul contenuto
  - Un grafico che mostri gli argomenti relativi ai contenuti più visti
  - Un grafico che mostri il profilo dei contatti associato ai contenuti più visti
  - Al click su un contenuto di questo grafico si apra un popup che contenga
    * Una descrizione del contenuto
    * Il numero di impression$_G$
    * Il numero di click$_G$ TODO impression e click o clickfrureit?
    * I tag del contenuto
    * Le tipologie degli utenti che hanno interagito col contenuto
    * Gli argomenti più trattati da queste tipologie di utenti

- Dare la possibilità di filtrare per periodo qualunque

- Dare la possibilità di cercare specifici contenuti

- Dare la possibilità di ordinare per più visti o più cliccati

**Opzionali:**

- Sviluppare una sezione in cui vengono dati consigli sulle raccomandazioni, ad esempio consigliare quali contenuti si consiglia di togliere dalle possibili raccomandazioni

- Sviluppare un meccanismo per la personalizzazione dei widget di raccomandazione

# Tecnologie utilizzate

# Attività svolte

## Prima fase

### Analisi tracciamenti

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

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

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

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

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

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

Mentre è facile capire che i contenuti più proposti siano i contenuti caricati più volte dai widget di raccomandazione (ovvero contenuti caricati nel widget nella pagina web), è importante capire la differenza tra visto e cliccato. Per contenuto visto (impression) si intende un contenuto proposto dal PCR e che è entrato nella viewport$_G$ dell'utente finale (ma che non necessariamente l'utente ci ha interagito). Per contenuto cliccato si intende un contenuto visto dall'utente in più su cui quest'ultimo ha fatto un click per aprirlo.
Infatti i widget di raccomandazione contengono una lista di contenuti con una thumbnail$_G$ e una breve descrizione, per aprire il contenuto vero e proprio l'utente può cliccarci sopra. Figura x TODO img pcr
Durante l'analisi si è capito che la sola azione di caricare il contenuto dal widget (evento load) non portava necessariamente informazioni molto utili poiché se un contenuto non è neanche entrato nella viewport dell'utente si può non contare proprio così abbiamo deciso di tracciare questo evento per avere più informazioni magari utili in futuro ma per ora ci concentreremo più sugli altri due eventi.
Abbiamo infatti, dopo brainstorming e ricerche in internet, trovato una metrica che dasse un indicazione soddisfacente dell'efficacia di un contenuto, questa metrica si chiama **click through rate**. In pratica è un valore calcolato dal rapporto del numero di interazioni con il contenuto con il numero di viste (clicks diviso impressions). Questo indice può assumere valori compresi tra zero e uno, in particolare un valore vicino all'unità sta ad indicare che il contenuto è stato aperto molte volte rispetto alle visite ricevute. Questo fatto indicherebbe che il contenuto raccomandato è interessante per quasi tutti gli utenti a cui è stato proposto indicando un ottimo lavoro da parte del motore di raccomandazione. Valori invece molto bassi di questo indice significherebbero che il contenuto è stato aperto poche volte rispetto al numero di visite, suggerendoci che il contenuto ha provocato scarso interesse da parte degli utenti a cui è stato proposto.
Si è deciso quindi che questo fosse uno dei valori, per ogni contenuto, da presentare al cliente.
Abbiamo anche pensato di eliminare casi particolari come gli elementi con un numero di visite al di sotto di una certa soglia. Questo perché uno scarso numero di visite e pochi click avrebbe comunque un click through rate più alto di contenuti magari più significativi. Per esempio un contenuto con 3 click e 4 impression avrebbe un click through rate di 0.75, indice molto alto, mentre un contenuto con 1000 impression e 500 click avrebbe un click through rate di 0.5, il che porterebbe a pensare che il contenuto sia molto meno interessante ma è probabile che in realtà sia il contrario perché con numeri molto bassi è facile avere un rapporto alto e non veritiero.
Queste considerazioni poi possono essere estese non solo a contenuti caricati

dal widget di raccomandazione ma bensì per qualunque altro ambito in cui un contenuto è condiviso e che si possa raccogliere dati di intelligence. Tutte queste considerazioni valgono non solo per contenuti caricati dai widgets di raccomandazione ma per qualunque contenuto THRON messo in pagina. Questo è stato motivo di ulteriore stimolo poiché erano funzionalità aggiunte per ora sullo specifico caso, ma in futuro estese ad altri progetti dell'azienda.

### Progettazione e sviluppo prima fase

#### Raccolta degli eventi - libreria dei tracciamenti

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

- **grunt:** è un javascript task runner, ovvero una libreria per automatizzare azioni che si eseguono spesso. Nel mio caso l'ho utilizzato per automatizzare la compilazione, la minimificazione e l'esecuzione di batterie di test
- **Lodash:** è un evoluzione di **underscore.js**, si tratta di una libreria che contiene moltissime funzioni di utilità con il massimo delle performance
- **ajax:** per effettuare chiamate http

e molte altre di minore importanza o per aggiungere il supporto di funzionalità altrimenti non supportate da alcuni browser, ad esempio per utilizzare le promises$_G$ su Internet Explorer 9.
L'inizio è stato difficile, dovevo capire perfettamente come funzionasse la libreria dei tracciamenti di THRON e poi capire come funzionasse e come fosse il workflow di Snowplow che è utilizzato all'interno della libreria per inviare i dati. La libreria infatti è di una certa complessità e non avevo mai dovuto studiare un modulo così grande durante i miei studi universitari.

Tuttavia la libreria era scritta molto bene, documentata e presentava delle pagine di demo che aiutavano molto a capire e a provare tutte le funzionalità. L'aggiunta dei nuovi eventi impression e click è stata semplice infine. Utilizzando l'ereditarietà prototipata di javascript ho creato un oggetto che ereditasse tutte le funzionalità già presenti e ho aggiunto due nuovi metodi per esporre al client la possibilità di tracciare questi eventi. Vista la natura di snowplow è stato necessario anche aggiungere degli schemi perché i nuovi tracciamenti avevano bisogno di inviare anche nuove informazioni come l'identificativo della applicazione di raccomandazione.
Questo era sufficiente per gli obiettivi riguardanti l'aggiunta degli eventi nella libreria dei tracciamenti, tuttavia per effettuare il tracciamento vero e proprio era ancora necessario che qualcuno scrivesse qualche riga di codice in tutte le pagine dove era necessario tracciare questo nuovo tipo di evento. Inoltre il codice per capire quando un contenuto entri nella viewport$_G$ di un utente non è affatto banale e non era accettabile che in ogni pagina si dovesse aggiungere queste cose.
Era quindi necessario creare un sistema che tracciasse in automatico, secondo logiche prefissate, contenuti THRON in pagine web senza l'ausilio di aggiunta di codice.
Il problema è molto ampio e non banale, tuttavia questo stage rappresentava un buon punto per iniziare.
I widget raccomandativi, come visto in precedenza TODO LINK, aggiungono in pagina una lista di immagini con descrizione. Queste immagini, chiamate thumbnail$_G$, rappresentano effettivamente il contenuto stesso quindi si è scelto di partire con un software che tracciasse in automatico eventi sulle immagini, lasciandolo aperto a future estensioni per tutti gli altri tipi di contenuto che, spesso, useranno le stesse logiche applicate per le immagini.
Ho pensato quindi di creare un modulo all'interno della libreria dei tracciamenti che usasse le funzionalità esposte dalla libreria stessa.
L'idea che ho avuto è stata creare un *parser*, ovvero un oggetto che, al caricamento della pagina o tramite invocazione javascript, analizzasse la pagina web e cercasse tutti gli elementi possibili da tracciare.
Due problemi erano presenti: come far capire al parser quali erano gli elementi tracciabili e come recuperare le informazioni per il tracciamento, infatti la libreria dei tracciamenti espone un oggetto chiamato *tracker* su cui invocare gli eventi da tracciare. Quest'ultimo necessità di informazioni come l'identificativo del contenuto, il codice servizio$_G$, il tipo del contenuto e altro.
Il mio tutor mi ha lasciato libera scelta per il primo problema e mi ha indirizzato invece su una facile soluzione per il secondo.
Mi ha mostrato che le thumbnail$_G$ usate per rappresentare i contenti nelle pagine web avevano un url costruito secondo un pattern ben definito, ovvero:

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

Questo mi ha eliminato il problema riguardante il come trovare le informazioni da passare al tracker$_G$ della libreria, da questi url infatti è facile estrarre il codice servizio, l'id del contenuto e il tipo di contenuto è sempre un immagine.
Rimaneva da trovare una soluzione per far capire al parser quali elementi in pagina erano da tracciare e quali azioni tracciare su di essi. Infatti nel caso del mio progetto bisogna tracciare gli eventi load, click e impression, tuttavia il modulo aggiuntivo è stato fatto pensando a sviluppi futuri perciò deve essere data la possibilità ai clienti di scegliere quali azioni tracciare su quali contenuti, non necessariamente tutti i contenuti THRON inclusi in pagina.
Ho scelto di risolvere entrambi questi problemi attraverso l'uso di classi CSS. L'idea è quella di aggiungere una classe, personalizzabile, per far capire al parser quali elementi devono essere analizzati e quali azioni tracciare in automatico su di essi.
Ho analizzato il problema col tutor e ha accettato la mia idea per la semplicità e perchè non rendeva necessaria nessuna modifica a servizi come quello che genera le thumbnail$_G$ che avrebbe causato ritardi nella realizzazione.
Ho così iniziato a sviluppare questo modulo nella libreria, mantenendo una struttura ad oggetti per lasciare aperte future estensioni per il supporto di nuovi tipi di contenuti o nuovi eventi. Su suggerimento del tutor ho aggiunto dei controlli per filtrare eventi non possibili su alcuni tipi di contenuto, come tracciare eventi seek$_G$ su immagini e per limitare gli errori possibili di battitura sulle classi.
L'idea che avevo avuto inizialmente era di sviluppare questo parser utilizzando tutte le funzionalità esposte dal framework jQuery, questo infatti mi avrebbe aiutato molto per la ricerca degli elementi, per il filtraggio e per avere un supporto multi browser assicurato. Tuttavia mi è stata sconsigliata perché è una libreria molto grande e avrebbe appesantito l'intera libreria. Ho così sviluppato tutto il modulo in puro javascript senza l'ausilio di altri framework eccetto quelli che già erano nella libreria.
Il supporto ad alcuni browser, in particolare Internet Explorer 9, mi ha dato qualche problema che ho dovuto risolvere con dei fallback o aggiungendo dei piccoli polyfill$_G$ che potranno essere utili anche in sviluppi futuri.
Ho reso poi il parser parametrico, utile in particolar modo la possibilità di scegliere di analizzare solo pezzi della pagina web identificabili tramite HTML id.

Ho creato inoltre delle pagine di demo apposite piene di immagini con tutti i casi limite a cui ho pensato per provare il modulo e per rendere anche più facile la comprensione per chi dovrà mantenerlo, mi sono ispirato infatti alle pagine di demo già presenti nella libreria dei tracciamenti che mi hanno aiutato moltissimo nella comprensione del software.
Il parser inoltre non analizza soltanto il codice HTML nel momento della sua instanziazione ma attraverso l'uso di mutation observers$_G$ continua ad ascoltare per cambiamenti nel DOM$_G$. Se ne avvengono nella parte di HTML che è stato istruito di analizzare controlla se ci sono cambiamenti delle classi o se vengono aggiunti elementi in pagina in un secondo momento, questo ultimo fattore rende tutto il tracciamento automatico senza aggiunta di codice in pagina come richiesto, eccetto una riga per istanziare il parser che è inevitabile.
Ho poi aggiunto le logiche per automatizzare i tracciamenti. Per l'evento load la logica era molto semplice: load sta a significare che il contenuto è stato caricato in pagina, perciò il parser non appena trova un contenuto che nella classe gli dice di tracciare l'evento load lo lancia.
Per il click è similmente facile, basta aggiungere un listener$_G$ all'elemento HTML che al click dell'utente fa partire l'evento.
Un altra storia è per l'evento impression: questo evento deve essere lanciato quando l'immagine in pagina entra nella viewport$_G$ dell'utente. Ho cercato molto online, sono presenti diverse librerie che cercano di risolvere il problema ma tutte hanno diversi problemi quindi ho sviluppato io la logica.
L'idea di base è aggiungere dei listener$_G$ allo scroll della pagina e fare dei controlli sulla posizione degli elementi rispetto alla pagina, il tutto era già fatto dagli script che si possono trovare con una piccola ricerca online ma il problema diventa più complesso quando si iniziano a considerare altri aspetti come lo scrolling orizzontale e il fatto che un elemento può essere all'interno di un DIV$_G$ anch'esso scrollabile.
Ho quindi creato una funzione che analizza la posizione di un elemento rispetto alla pagina e rispetto al primo e al secondo elemento HTML scrollabile tra i suoi ascendants$_G$ nel caso ci siano.
Ho scelto di non andare oltre al secondo livello di parentela per due motivi:

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

Ho provato il modulo sulle mie pagine di test contenenti più di cinquecento immagini con DIV$_G$ con tutti gli scroll possibili e il parser sembrava andare

molto bene e non ho notato nessun tipo di rallentamenti.
Il tutor ha approvato il lavoro fatto e il parser è stato anche messo in alcune pagine del sito di THRON per testarlo in un caso d'uso vero dandomi molta soddisfazione.
Per aggiungere la raccolta automatica dei widget di raccomandazione mi è bastato aggiungere nel codice del widget stesso le istruzioni per istanziare un parser e per far partire il parsing della sezione HTML in cui il widget è contenuto, questo infatti è associato ad un id che il widget conosce.
L'analisi e la progettazione di queste attività sono state più lunghe del previsto, si è deciso quindi di cercare di risparmiare tempo nelle prossime attività qual'ora ci fosse l'opportunità, altrimenti di tralasciare qualcuno degli obiettivi non obbligatori come lo strumento di personalizzazione dei widget.

**Elaborazione dei dati raccolti - Spark**
Come descritto in precendeza, in TODO LINK, il workflow di Snowplow prevede che i dati vengano raccolti in un posto chiamato *collector*$_G$, qui si troveranno tutti i dati dei tracciamenti grezzi, così come sono inviati dai client. Per renderli presentabili all'utente finale hanno bisogno di alcune elaborazioni che sono fatte da degli *enricher*$_G$.
THRON ha sviluppato diversi *enricher*$_G$ per diversi bisogni, quattro in particolare:

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

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

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

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

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

## Seconda Fase

### Analisi raccolta del numero di raccomandazioni

Ho partecipato a delle riunioni col team Core per capire come rendere disponibili ai clienti i dati relativi alla quantità di raccomandazioni fatte dalle loro applicazioni. Questo è molto importante poiché i clienti al momento dell'acquisto dello strumento di raccomandazioni possono scegliere diversi Business Model che prevedono una quantità di raccomandazioni oraria massima prefissata. Superata questa soglia lo strumento non raccomanda più secondo

algoritmi intelligenti ma propone solo contenuti simili.
THRON possiede un Elasticsearch$_G$ in cui vengono immagazzinati tutti gli accessi alle loro API. Tuttavia questi dati vengono salvati per un massimo di sessanta giorni poiché la mole di dati è non indifferente (si parla di almeno quattro gigabyte di log di accessi al giorno contando solo gli accessi alle API, senza contare quindi le fruizioni delle loro cdn$_G$).
I dati da fornire ai clienti devono avere una retention$_G$ di almeno due anni, per questo motivo si è deciso di creare un qualche processo che ad intervalli regolari andasse ad interrogare l'Elasticsearch di monitoring, ricavasse tutte le informazioni riguardanti gli accessi alle raccomandazioni e le andasse a salvare in qualche altro database.
I grafici potranno avere una granularità oraria, giornaliera o mensile, si è scelto che i dati debbano essere salvati raggruppati per queste granularità per avere un risultato immediato.
L'Elasticsearch di cui si parla è in una rete privata dell'azienda perché contiene dati che non devono essere resi pubblici, analogamente il database in cui si andranno a salvare i dati sarà nella stessa rete interna è quindi necessario anche creare uno strato di web services che rendano disponibili i dati riguardanti la quantità di raccomandazioni all'esterno.
Come ultima cosa abbiamo pensato anche di creare, nel caso di fosse tempo, un servizio che ripulisse i dati più vecchi di due anni con granularità maggiore di quella mensile. Questo perché è poco auspicabile che un utente voglia vedere le raccomandazioni relative ad un giorno preciso di più di due anni fa.
Inoltre la quantità di dati da salvare non è molto grande ma si parla comunque di una riga su un database per ogni ora per ogni applicazione raccomandativa, facendo dei calcoli si ottengono **8760** entry orarie per applicazione. Non è un numero molto grande ma bisogna pensare che le applicazioni raccomandative potrebbero aumentare di molto andando a aumentare l'ordine di grandezza dei dati da salvare.

### Progettazione e sviluppo seconda fase

Per prima cosa mi è stato chiesto di pensare alle firme di tutti i servizi da creare. THRON mantiene tutta l'architettura software in un progetto di *Enterprise Architect* tramite diagrammi delle classi e altro ancora.
Hanno sviluppato un plugin che a partire dai diagrammi crea tutte le interfacce scala automaticamente.
Mi hanno dato l'accesso a diverse loro repositories$_G$ per farmi familiarizzare con il loro lato server, questo mi ha aiutato molto per la stesura delle firme dei vari metodi, anche perché la serie di web services per l'interrogazione dei

reports già esistente era comunque simile a ciò che dovevo fare io. Sfruttando anche l'ereditarietà e i mixin Scala ho potuto riutilizzare parte del loro codice anche per le firme.
Create e concordate le firme il team Core le ha aggiunte alla loro infrastruttura e ha generato le interfacce. Ho dovuto quindi creare le implementazioni di quest'ultime e per la parte di web services ho seguito molto quelli già esistenti. Questo mi ha facilitato molto il lavoro, ho potuto riutilizzare molti componenti di utilità già scritti da THRON come i servizi per l'autenticazione e dei meccanismi per l'accesso al database.
Si è scelto di usare un MongoDB$_G$ per il salvataggio dei dati per due motivi:

- Il team Core faceva già un grande uso di questo database, c'erano già quindi classi di utilità per l'utilizzo di quest'ultimo
- I dati da salvare non erano di tipo relazionale quindi era più idoneo usare un database NoSQL

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

- Per prima cosa legge dal database quando è stata l'ultima elaborazione dei dati
- Interroga Elasticsearch chiedendo i dati relativi alle raccomandazioni dall'ultima elaborazione
- Crea un documento, nella collezione contenente i dati orari, con al suo interno il numero di raccomandazioni fatte per ogni applicazione raccomandativa
- Controlla se è passato più di un giorno dall'ultima elaborazione dei dati giornaliera
- Se si raccoglie, al suo interno, i dati relativi all'ultimo giorno e salva in una collezione adibita a contenere i dati giornalieri un documento con quest'ultimi

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

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

# Terza fase

## Analisi esposizione dati

Lo scopo di questa fase era trovare dei grafici adatti ad esporre in una dashboard i dati raccolti in maniera chiara e concisa.
Sono state fatte delle riunioni con membri del team di Design e di UX sia per capire quali grafici inserire nella dashboard, l'usabilità che essa deve fruire e come integrarla in bacheca$_G$.
Mi sono state imposte delle tecnologie per esempio l'integrazione in bacheca

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

## Progettazione e sviluppo terza fase

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

- D3js
- FusionCharts
- KendoUI

**D3js:** (Data Driven Documents) è una libreria gratuita nata nel 2011 per la visualizzazione dinamica e interattiva di dati organizzati, offre funzionalità a basso livello per creare grafici attraverso oggetti SVG.

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

struttura analoga alla loro.
Credevo di avere più problemi per il grande supporto a browser richiesto ma grazie all'uso di molti framework e librerie non ho avuto problemi di questo tipo.
Alla fine di queste attività era finito l'intero periodi di stage per cui non è stato possibile realizzare il meccanismo per la personalizzazione dei widgets di raccomandazione.

# Obiettivi raggiunti

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

**Obbligatori:**

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

- L'enricher$_G$ dovrà salvare tutti gli eventi in indici diversi raggruppati per tipo di evento
- Aggiungere al modulo dei Reports$_G$ il supporto ai nuovi eventi
- Far si che gli altri moduli di Intelligence$_G$ ignorino questi nuovi eventi per il momento

**Desiderabili:**

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

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

**Obbligatori:**

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

**Desiderabili:**

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

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

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

**Opzionali:**

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

# Old stuff

### Tracciamento eventi - libreria dei tracciamenti

Inoltre si è scelto di aggiungere alla libreria la logica per tracciare questi eventi in automatico, cioè senza aggiunte di codice lato client in pagina, per facilitare ulteriormente i tracciamenti non solo per il widget$_G$ raccomandativo ma anche per qualunque altra pagina web che necessiti di tracciare eventi come click, load, impression o altri eventi che verranno aggiunti in futuro.
Questo ha provocato una modifica al mio piano di lavoro immettendo qualche giorno di lavoro (stimati cinque) per studiare la libreria dei tracciamenti, le tecnologie utilizzate, progettare come apportare le modifiche e svilupparle prima dello sviluppo del resto del progetto.

### Raccolta ed elaborazione dati tracciamenti

La libreria dei tracciamenti di THRON utilizza snowplow$_G$, una libreria atta appunto a tracciare eventi anche personalizzati. Grazie a questa tecnologia tutti i dati relativi ai tracciamenti restano proprietari dell'azienda al contrario di altri servizi di tracciamenti come Google Analytics. Il vantaggio di snowplow è sicuramente il fatto di essere proprietari di tutti i dati tracciati e la totale personalizzazione che si può fare, sia negli eventi da tracciare sia nelle fasi successive di elaborazione di questi dati. Il lato negativo è che bisogna scriversi tutto il codice necessario per gli eventi personalizzati e scrivere il codice dei software che ricevono i tracciamenti, che gli elaborano e che li mettono a disposizione di eventuali client.
I dati tracciati dalla libreria javascript presente in tutte le pagine necessarie vengono raccolti in un collector che può essere visto come un calderone in cui arrivano e vengono raggruppati tutti gli eventi tracciati. Questi dati vengono letti da un enricher, realizzato con Apache Spark$_G$, che si occupa di arricchirli con altre informazioni, ad esempio viene effettuato il merge$_G$ dei contatti, questo va a rendere possibili filtraggi molto mirati e analisi di dati per strategie business impossibili altrimenti.

I dati a questo punto vengono elaborati ulteriormente da diversi enricher (altre istanze Spark) ed ognuno aggiunge informazioni utili per diversi scopi. Al momento ne hanno tre:

- Recommendation: questo contiene tutta la logica per la raccomandazione dei contenuti e l'aggiunta di importanti informazioni
- Behaviour: questo serve per analizzare il comportamento degli utenti
- Reports: questo serve per arricchire con moltissimi dati gli eventi per rendere possibile la fruizione di informazioni utili per business strategy

Tutte queste istanze salvano poi i dati in diversi Elasticsearch che possono essere interrogati da web services appositi dall'esterno.
Andando ad aggiungere nuovi eventi è necessario modificare il primo enricher comune per aggiungere il support a questi nuovi eventi. Per i tre enricher specifici invece è necessario effettuare queste modifiche:

- Recommendation: filtrare i nuovi eventi, per ora gli algoritmi di raccomandazione non utilizzeranno impressions e interacts. Questa è una modifica che sarà effettuata nel prossimo futuro poiché non banale da applicare. Infatti il recommendation utilizza algoritmi di machine learning per essere più preciso ed efficiente
- Behaviour: anche in questo caso filtrare i nuovi eventi, come per il recommendation sono modifiche da applicare con molta cautela in algoritmi delicati e lo stage non prevede abbastanza tempo
- Reports: qui al contrario bisogna fare in modo che i nuovi eventi vengano letti, elaborati ed arricchiti con tutte le informazioni necessarie. Inoltre i dati che vengono salvati su Elasticsearch è necessario indicizzarli raggruppandoli per tipo di evento per maggiore ordine

I web services per ricavare questi di dati verranno sviluppati da altri membri del team di Intelligence essendo presenti già servizi molto simili sono modifiche che applicheranno molto velocemente.

## Analisi fase 2

Successivamente ho partecipato a delle riunioni col team Core per capire come rendere disponibili ai clienti i dati relativi alla quantità di raccomandazioni fatte dalle loro applicazioni. Questo è molto importante poiché i clienti al momento dell'acquisto dello strumento di raccomandazioni possono scegliere

diversi Business Model che prevedono una quantità di raccomandazioni massima prefissata. Superata questa soglia lo strumento non raccomanda più secondo algoritmi intelligenti ma propone solo contenuti simili.
THRON possiede un Elasticsearch$_G$ in cui vengono immagazzinati tutti gli accessi a tutte le loro API. Tuttavia questi dati vengono salvati per un massimo di sessanta giorni poiché la mole di dati è non indifferente (si parla di almeno quattro gigabyte di log di accessi al giorno contando solo gli accessi alle API, senza contare quindi le fruizioni delle loro cdn$_G$).
I dati da fornire ai clienti devono avere una retention$_G$ di almeno due anni, per questo motivo si è deciso di creare un qualche processo che ad intervalli regolari andasse ad interrogare l'Elasticsearch di monitoring, ricavasse tutte le informazioni riguardanti gli accessi alle raccomandazioni e le andasse a salvare in qualche altro database.
I grafici potranno avere una granularità oraria, giornaliera o mensile dovranno essere quindi salvati i dati raggruppati per queste granularità per avere un output immediato.
Per rendere poi questi dati accessibili all'esterno della rete aziendale è necessario anche creare dei webservices da aggiungere a quelli già esistenti in THRON. Questo lavoro mi era di molto facilitato grazie al fatto che ci fossero già molti altri webservices, infatti potevo riutilizzare molti componenti già esistenti come il meccanismo di autenticazione e meccanismi di accesso ai database.
Essendo la quantità di dati elevata di è deciso, qual'ora il tempo lo permettesse, di aggiungere anche un servizio che andasse a ripulire i dati più vecchi di due anni con granularità maggiore di quella mensile.

## Analisi fase 3

Infine sono state fatte le analisi per discutere come presentare questi dati ai clienti.
col team di design si sono scelti i tipi dei grafici da presentare per ottenere una maggior usabilità possibile e per rendere i dati di facile comprensione, cosa non banale avendo moltissime informazioni da esporre.
Col team UX si è discusso invece di come integrare questi grafici nel loro software in cui sono già presenti molti grafici per la presentazione di informazioni di Intelligence. TODO aggiungere img grafici intelligence

## Tecnologie utilizzate

### Javascript

Javascript è un linguaggio di scripting orientato agli oggetti e agli eventi. Principalmente è utilizzato per la programmazione lato Client negli ambienti Web per aggiungere funzionalità, effetti dinamici o interattività alle pagine o applicazioni web.
Da qualche tempo è però utilizzato in molti altri ambiti come la programmazione di applicazioni mobile tramite frameworks come ionic o appcelerator.
Essendo Thron una piattaforma web Javascript è un linguaggio pressoché obbligatorio da imparare per lavorare in azienda.
Nel mio stage l'ho utilizzato largamente per apportare le modifiche alla libreria dei tracciamenti visto che quest'ultima è interamente una libreria realizzata in javascript.
Ho utilizzato diverse librerie come **Lodash**, **CommonJS**, **snowplow**, **jQuery**, **Bootstrap** e **grunt**.
Inoltre ho utilizzato javascript anche per l'esposizione dei dati, ovvero per la creazione della console in cui sono esposti dei grafici come spiegato nei capitoli a seguire. Per questi l'azienda mi ha dato piena facoltà di scegliere e decidere cosa usare così ho studiato e provato le tre librerie che mi sembravano più adatte: **FusionCharts**, **KendoUI** e **d3js**.

FusionCharts è una libreria a pagamento nata nel 2003 e offre una vasta gamma di diagrammi, mappe, widgets e dashboards già fatti con cui arricchire i propri siti o applicazioni web. Dopo averla provata è risultata essere molto facile da imparare e immediata all'uso, quasi non necessita di conoscenze di programmazione. Molto belli e veloci i risultati che si possono ottenere tuttavia meno personalizzabili delle altre librerie provate proprio per la sua grande semplicità. Un altro punto a favore è la retro compatibilità molto estesa e resa automatica dalla libreria.
D3js, che sta per data driven documents, è un altra libreria nata nel 2011 per la visualizzazione dinamica e interattiva di dati organizzati, quindi offre funzionalità a basso livello per creare grafici. Si può arrivare a dei risultati sbalorditivi e davvero di forte impatto, tuttavia a me non servivano diagrammi particolarmente complessi e in più necessita di molto più lavoro per ottenere buoni risultati. In questo caso la retro compatibilità non è sempre garantita per le versioni dei browser non recenti.
KendoUI è un framework che offre una grandissima quantità di funzionalità built-in ma io mi sono concentrato nelle API per creare grafici. Anche in questo caso molto semplice da usare, risultati immediati e molto belli, in più

offre più personalizzazione e una buona retro compatibilità.
La scelta è ricaduta su FusionCharts poichè molti grafici già realizzati da THRON sono creati usando questa libreria, quindi risultava essere tutto più omogeneo utilizzando la stessa, inoltre essendo un pò stretti con i tempi l'uso di FusionCharts ho pensato potesse aiutare a farmi rientrare nei tempi.

### HTML e CSS

Naturalmente per la creazione dei grafici e l'integrazione in bacheca$_G$ ho utilizzato anche questi linguaggi.
Le ho utilizzate anche per creare delle pagine di demo e di test per il parser$_G$ creato per effettuare i tracciamenti in automatico.

# Progettazione e sviluppo

# Chapter title

## Section title

Sed varius rhoncus libero a consequat. Cras facilisis magna eget tellus laoreet sit amet mattis nulla posuere. Nullam magna est, porta a feugiat quis, mollis vel urna. Nunc ante dolor, pretium eget laoreet in, tincidunt vitae ipsum. Nulla et diam risus. Ut auctor auctor vestibulum. Vestibulum vitae turpis sit amet lacus pulvinar dictum laoreet vitae enim. Aliquam erat volutpat. Aliquam ultricies posuere sem, ac mollis nunc interdum in. Vivamus tempor felis a tellus volutpat ut elementum lorem congue. Proin purus tortor, ultricies vitae viverra non, feugiat ac felis. Fusce condimentum dignissim volutpat. Proin quis augue ac tortor mollis congue at et magna. Nullam a velit est, nec ultrices justo (vedi Figura 5.1).




Figura 5.1: Image caption

### Sub-section title

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas (vedi Listing 5.1). Suspendisse arcu magna, faucibus ut tincidunt non, ultrices ut turpis. Nullam tristique vehicula massa, id commodo orci sollicitudin vel. Donec nibh ante, ultrices non facilisis sed, mattis id ligula. Sed sed orci sit amet nulla egestas gravida. Suspendisse laoreet, massa vel sagittis gravida, lectus ligula feugiat risus, a aliquam dolor eros ac orci. Nulla egestas tortor quis nunc scelerisque sed tincidunt massa scelerisque. Pellentesque vulputate pharetra lectus, vitae ultricies nisi luctus eu. Nam congue dui eu quam euismod vitae fermentum sem vehicula. Etiam ac leo id nisi placerat posuere. Curabitur mattis augue eget dolor tempus accumsan consequat diam imperdiet. Sed tristique orci id lacus vulputate rhoncus. Morbi tincidunt ante sed turpis luctus tincidunt et sit amet augue. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Nunc viverra urna non libero sodales euismod et eleifend sapien. Donec aliquet risus non massa dignissim sollicitudin. Integer a ligula eros. Morbi et lacinia augue [**?**].

```
<p>
Pellentesque ac tortor eget eros iaculis euismod
vitae vitae augue.
</p>
<!— comment —>
```

Listing 5.1: caption text