

Università degli Studi di Padova

DIPARTIMENTO DI MATEMATICA "TULLIO LEVI-CIVITA"

CORSO DI LAUREA IN INFORMATICA



Una base di dati a grafo per una applicazione antifrode

Tesi di laurea triennale

Relatore

Prof. Tullio Vardanega

Laureando

Luca Dario

ANNO ACCADEMICO 2016-2017

The only way to do great work is to love what you do. If you haven't found it yet,
keep looking. Don't settle.

— Steve Jobs.

Sommario

Il presente documento descrive il lavoro svolto durante il periodo di stage dal laureando Luca Dario presso l'azienda XTN – Cognitive Security S.r.l di Padova (PD). Lo stage è stato svolto alla conclusione del percorso di studi della Laurea Triennale ed è durato in totale 316 ore.

Gli obiettivi erano focalizzati nel condurre una analisi sulle tecnologie di basi di dati a grafo, al fine di risolvere un noto problema all'azienda. Infine era richiesto di produrre un prototipo che esemplifichi le capacità relazionali e prestazionali delle tecnologie scelte.

I primi due capitoli del documento hanno lo scopo di presentare il contesto aziendale in cui è stato sostenuto lo stage e di presentare il progetto richiesto. Il terzo capitolo documenta lo svolgimento dello stage descrivendo le attività fondamentali svolte per soddisfare gli obiettivi posti dall'azienda. Infine il quarto capitolo presenta una valutazione dello svolgimento dello stage rispetto a conoscenze acquisite ed obiettivi aziendali.

*“The consciousness of loving and being loved brings a warmth and a richness to life
that nothing else can bring.”*

— Oscar Wilde

Ringraziamenti

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

Desidero ringraziare con affetto sia i miei genitori che mio fratello per il sostegno, economico e morale, e per essermi stati vicini in ogni momento durante gli anni di studio.

Vorrei infine ringraziare tutti gli amici che mi sono stati a fianco durante questi anni di studio. In particolar modo Riccardo, Stefano, Nicolò, Alberto e Lisa. Oltre a loro, tengo a ringraziare anche tutti gli altri che mi hanno aiutato ad arrivare dove sono.

Padova, Dicembre 2017

Luca Dario

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Il contesto aziendale

1.1 L'azienda: XTN – Cognitive Security

XTN – Cognitive Security nasce nel 2013 come azienda focalizzata nello sviluppo di soluzioni basate sui profili comportamentali, cioè l'analisi dei comportamenti abituali di determinati utenti, in ambito di sicurezza ed anti frode. Nel 2014 ha rilasciato le prime soluzioni sulla protezione ed analisi delle transazioni finanziarie.

XTN è situata in tre diverse sedi nel nord Italia, Padova, Milano e Rovereto. Attualmente è attiva nello sviluppo di soluzioni **B2B** sia in ambito anti frode che di applicazioni per la protezione di dispositivi mobili e *IoT*.

1.2 Prodotti offerti

XTN – Cognitive Security offre 2 prodotti di punta, **Smash®**¹ e **More®**².



Figura 1.1: Logo *Smash®* (<https://xtn-lab.com/smash/>)

Smash® è un **framework** che analizza il comportamento abituale, grazie a più di cento parametri, degli utenti di servizi di pagamento online. Grazie a questa funzionalità riesce a stabilire, in tempo reale, il fattore di rischio di ogni transazione.

Il fattore di rischio è calcolato attraverso svariati algoritmi comportamentali personalizzabili dall'utente tramite un apposito *editor* integrato alla piattaforma. Se, grazie a questi algoritmi, una transazione dovesse risultare sospetta, questa verrà notificata ad una persona incaricata a verificarne l'effettiva natura.

Non vengono analizzati solamente i parametri delle transazioni, ad esempio il destinatario o la geolocalizzazione, ma anche fattori esterni come anomalie nei protocolli di

¹Smash: url= <https://xtn-lab.com/smash/>

²More: url= <https://xtn-lab.com/more/>

comunicazione, manipolazioni HTML, compromissioni del dispositivo mobile. Questo prodotto è destinato a istituti finanziari, negozi online e assicurazioni.



Figura 1.2: Logo More® (<https://xtn-lab.com/more/>)

More® invece è un servizio installabile nei dispositivi mobili in grado di verificare se l'utente che sta utilizzando il dispositivo, in un determinato istante, coincide con l'utente che dice di essere. Questo è possibile analizzando molteplici parametri, tra cui lo stato del dispositivo, correlandoli poi con una analisi comportamentale e biometrica. More® è installabile in una qualsiasi applicazione mobile contenente informazioni confidenziali o critiche, come ad esempio un'applicazione bancaria. Tutte le operazioni necessarie per il funzionamento di More® sono eseguite in ambiente *cloud* preservando, quindi, le prestazioni dell'applicazione che lo ospita.

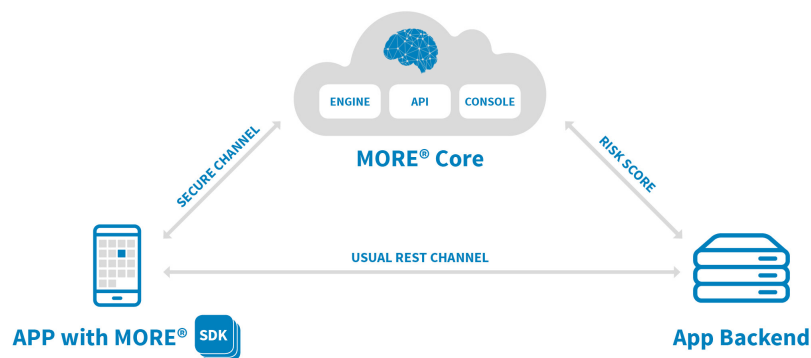


Figura 1.3: Architettura cloud More® (<https://xtn-lab.com/more/>)

1.3 Organizzazione aziendale

XTN adotta l'*Agile Team Organisation* tramite *Squads*, *Chapters*, *Guilds*.

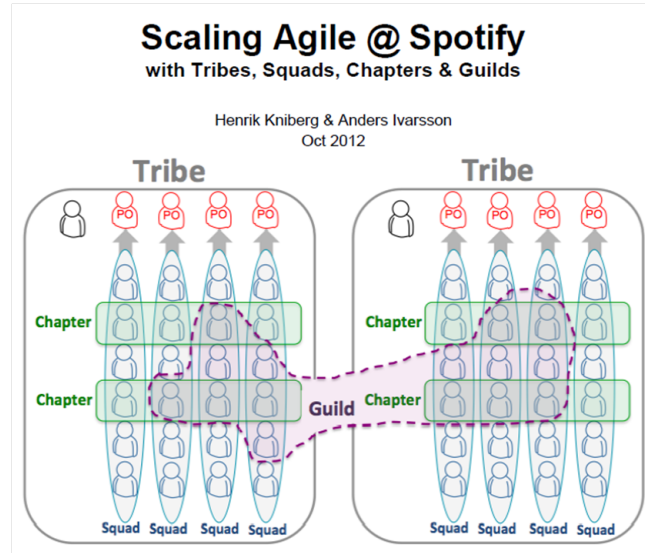


Figura 1.4: Agile Team Organisation (<https://goo.gl/fjJ006>)

- * **Squads:** sono un gruppo di persone che condividono il luogo di lavoro ed il campo di interesse. In XTN esistono due *squads*, uno per Smash® ed uno per More®. Ogni *squads* si riunisce periodicamente, circa ogni settimana, per tracciare e discutere la situazione delle varie *milestone*.
- * **Chapters:** sono un gruppo di persone che condividono le competenze, trasversalmente ai vari *squads*, per promuovere l'innovazione e la collaborazione. Periodicamente, circa ogni mese, i vari *chapters* si riuniscono per condividere informazioni di interesse e favorire un allineamento tecnologico tra i vari *squads*.
- * **Guilds:** sono una comunità di membri, in tutta l'organizzazione, che vogliono condividere conoscenze, strumenti e pratiche comuni. Ad esempio, in XTN, esiste una *guild* nata per accrescere la conoscenza dell'intelligenza artificiale.
- * **Tribe:** sono nate per suddividere, e rendere più gestibile, una grande infrastruttura. XTN non ha adottato questo concetto essendo ancora un'azienda di piccole dimensioni.

In particolare, durante il periodo di stage, sono stato localizzato nello *squads* di Smash® ma non appartenevo a nessuna *guilds*.

1.4 Processi aziendali e strumenti di supporto

In questa sezione verranno affrontati i processi aziendali e le tecnologie a supporto che ho avuto modo conoscere ed imparare lavorando a stretto contatto con il *team* di Smash®.

1.4.1 Gestione della configurazione

Tutti gli oggetti, che siano documentali e non, sono mantenuti in un *repository* interno. La configurazione è un insieme di questi oggetti che andranno a comporre una determinata versione del prodotto. Ogni configurazione ha il proprio identificativo univoco e un documento contenente informazioni utili all'installazione del prodotto.

Ogni oggetto per essere aggiunto ad una determinata configurazione deve essere prima verificato dall'amministratore di progetto.

Strumenti di supporto

Per tener traccia di tutte le modifiche ed evitare sovrascritture involontarie, l'azienda ha deciso di adottare una repository interna chiamata **Stash** ³. Quest'ultima è un sistema di gestione di *repository* che utilizza diversi sistemi di controllo di versione, tra cui Git e Mercurial. Infine l'azienda ha deciso di utilizzare **Git** ⁴ come controllo di versione.

1.4.2 Processo di sviluppo

L'azienda si ritrova spesso a dover vendere il proprio prodotto con un certo numero di personalizzazioni. Per far ciò gli addetti alla vendita di XTN effettuano incontri con i clienti per ricavare idee ed osservazioni per, successivamente, stilarli in un documento. Quest'ultimo poi verrà usato dal team di sviluppo per creare il prodotto in linea con le richieste del cliente.

L'azienda non rilascia protipi incompleti al cliente ma solamente prodotti funzionanti con tutte le personalizzazioni richieste.

Attualmente per ogni personalizzazione richiesta dal cliente, il team di sviluppo cerca di renderla una *feature* usabile da tutti e quindi rivendibile.

Strumenti di supporto

* Backend

Per quanto riguarda la parte logica l'azienda ha deciso di utilizzare principalmente **Java** ⁵ come linguaggio, in quanto è molto performante ed ha una sintassi molto regolare. Per quest'ultimo il team di sviluppo ha deciso di adottare il **framework Spring** ⁶ per astrarre, e semplificare, buona parte di architettura di *backend*.

Per quanto riguarda la persistenza dei dati, XTN, ha deciso di adottare **MongoDB** ⁷.

³Stash: url= <https://it.atlassian.com/>

⁴Git: url= <https://git-scm.com/>

⁵Java: url= <https://www.java.com/it/>

⁶Spring: url= <https://spring.io/>

⁷MongoDB: url= <https://www.mongodb.com/it>



Figura 1.5: Logo Java e Spring (<https://goo.gl/Gm7usK>)

* Frontend

Per quanto riguarda l'interfaccia grafica della *dashboard* di Smash[®], XTN ha deciso di adottare il **framework AngularJS 1**⁸. Quest'ultimo è sviluppato da Google e permette di semplificare lo sviluppo e *test* di applicazioni web *single page*.

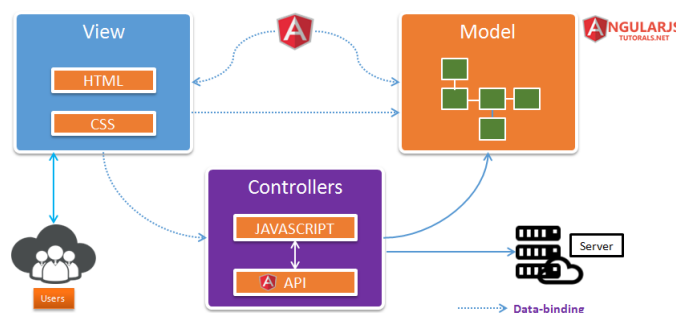


Figura 1.6: Architettura AngularJS (<https://goo.gl/PzB2v8>)

1.4.3 Processo di manutenzione

Ogni malfunzionamento viene segnalato e tracciato con la rispettiva priorità. Il personale addetto risolverà le varie segnalazioni in ordine di priorità e successivamente, dopo la sistemazione di un certo numero di malfunzionamenti, il *project manager* rilascerà la nuova versione del prodotto.

Strumenti di supporto

Qualunque malfunzionamento viene segnalato e registrato in un sistema di tracciamento delle *issues* chiamato **Jira**⁹. Questo servizio permette di assegnare, per ogni *issues*, una priorità, una scadenza e la versione con cui il malfunzionamento sarà sistemato.

1.4.4 Processo di verifica

Il processo viene gestito dal *chapter* che garantisce la qualità del prodotto offerto. Questo gruppo di persone esegue delle prove per garantire degli *standard*, in ambito di usabilità e prestazioni. Una volta verificate le varie parti il prodotto potrà essere messo in produzione.

⁸Angular 1: url= <https://angularjs.org/>

⁹Jira: url= <https://www.atlassian.com/software/jira>

1.5 Rapporto con l'innovazione

Attualmente i prodotti di *XTN – Cognitive Security* hanno raggiunto una maturità tale per cui il servizio che si promettono di offrire coincide con quello offerto realmente. L'azienda, però, continua periodicamente ad investire risorse per trovare nuove tecnologie o soluzioni al fine di offrire un servizio sempre migliore.

Queste risorse vengono divise tra studi di settore per focalizzarsi sulle tecnologie del momento e, successivamente, vengono svolti progetti interni, come stage universitari, per verificare la fattibilità della soluzione in analisi.

Proprio a questo riguardo, da circa un anno, è in corso uno studio della tecnologia di **apprendimento automatico** attraverso corsi di formazione e convegni. Questo viene svolto per riuscire ad aggiungere funzionalità che permettono l'apprendimento automatico del comportamento abituale di determinati utenti. Aggiungere questa funzionalità permetterebbe di abbassare nettamente i falsi positivi di frode bancarie.

Capitolo 2

Stage come tecnica aziendale

2.1 Descrizione del progetto

XTN – Cognitive Security è il primo anno che attiva progetti di stage in collaborazione con l'Università di Padova. Questo, però, non ha portato a ritardi o disagi per la mancata esperienza, da parte dell'azienda, in questo ambito.

2.1.1 Problema da risolvere per l'azienda

Tra le tante operazioni che Smash® esegue, esso calcola anche la **reputazione** che un determinato utente ha.

Questa reputazione si suddivide nei 2 seguenti tipi:

- * **Reputazione totale:** è il numero di transazioni bancarie che un determinato utente ha ricevuto.
- * **Reputazione relativa:** è il numero di transazione che un determinato utente ha ricevuto da un utente a scelta.

Il fattore di rischio, che Smash® calcola per ogni transazione, è dato anche dalla combinazione di questi 2 tipi di reputazione.

Fintanto che la reputazione viene calcolata su un'entità che riceve un numero modesto di transazioni giornaliere la velocità di esecuzione avviene in un tempo accettabile.

Il problema emerge quando questo calcolo viene eseguito su un'entità con un enorme numero di transazioni, come può essere un'azienda di dominio pubblico che riceve giornalmente transazioni. Questo avviene perché Smash® per calcolare la reputazione, deve caricare totalmente nella memoria il documento contenente tutte le transazioni di quella determinata azienda.

2.1.2 Possibile soluzione

Il problema potrebbe risolversi utilizzando un determinato tipo di base di dati organizzata a **grafo**.

Un grafo è un insieme di **nodi** che possono essere collegati tra loro attraverso delle relazioni chiamate **archi**.

Questi tipi di base di dati riescono a rappresentare le informazioni utilizzando solamente nodi e archi.

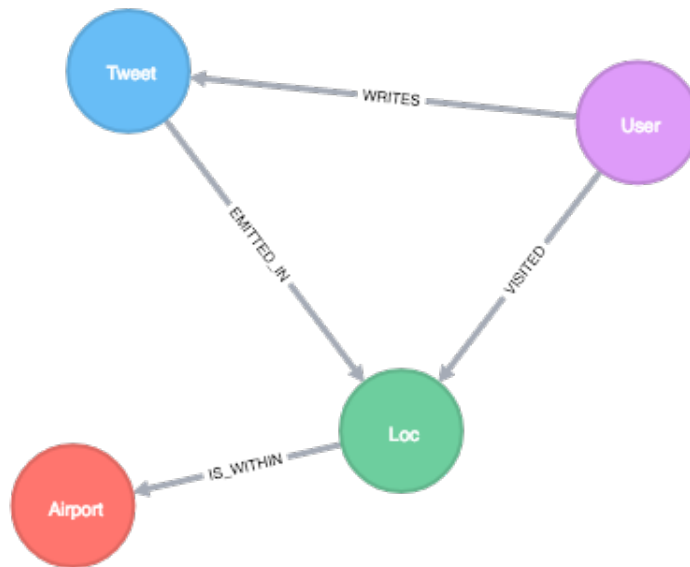


Figura 2.1: Esempio grafo orientato (<https://goo.gl/Z7S8J5>)

Un **nodo** può rappresentare un'entità fisica come ad esempio una persona o, in questo preciso contesto, un utente di una banca. Al contrario un **arco** rappresenta una relazione che avviene tra un nodo ed un altro, come ad esempio una transazione bancaria. Qualsiasi **nodo** ed **arco** possono avere un numero a piacere di proprietà, formate da una chiave univoca ed un preciso valore. Inoltre gli **archi** possono avere un verso di relazione ed un valore che rappresenta il tipo di relazione che l'arco descrive. **Se l'operazione che calcola la reputazione usasse una rappresentazione dei dati sfruttando la natura dei grafi, il calcolo della reputazione banalmente potrebbe ridursi a contare gli archi entranti di un determinato nodo.** Questo calcolo potrebbe non dipendere dal numero di transazioni che una determinata entità ha ricevuto.

Nella figura 2.2 *Luca* avrebbe una reputazione totale pari a 3 ed una reputazione relativa, rispetto a *Giovanni*, pari a 2.

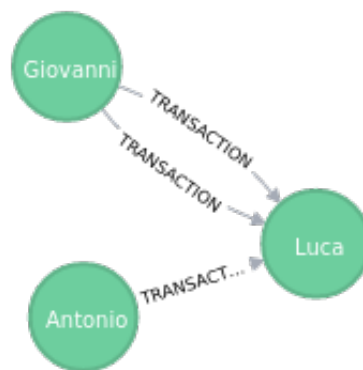


Figura 2.2: Possibile implementazione della base di dati

2.1.3 Direttive del progetto di stage

Il mio tutor aziendale, *Giuseppe Pavan*, mi ha incaricato di svolgere un'analisi sulle principali tecnologie che adottano una base di dati a grafo e , successivamente, sviluppare un prototipo che cercasse di risolvere il problema precedentemente descritto.

L'unico vincolo imposto dal mio tutor aziendale è l'utilizzo del linguaggio Java, visto l'uso massiccio che ne fa Smash®.

Gli obiettivi sono stati suddivisi dal mio tutor aziendale in due categorie: obbligatori e facoltativi.

Gli **obiettivi obbligatori** sono:

- * **Documentazione:** sviluppare un documento contenente la descrizione dettagliata del lavoro di valutazione e identificazione delle soluzioni disponibili in ambito delle base di dati a grafo. Il documento descriverà i vari *software* valutati, caratteristiche, vincoli e i razionali rispetto le scelte affrontate.
- * **Ambiente Docker:** predisposizione di un ambiente di lavoro, con le tecnologie scelte, tramite *container Docker*.
- * **Sviluppo prototipo:** sviluppo di una applicazione prototipale in Java che esemplifichi, partendo da un *dataset* di esempio, le capacità relazionali e prestazionali delle tecnologie scelte.

Gli **obiettivi facoltativi** sono:

- * **Estensione del dataset:** aggiungere maggiore complessità nel *dataset* d'esempio.
- * **Presentazione:** preparazione di una presentazione interna da esporre al *team* focalizzato sull'ambito anti frode.

2.1.4 Pianificazione

La pianificazione del progetto di stage l'ha eseguita il mio tutor aziendale nel seguente modo:

Giorni	Attività
3	Introduzione ad ambito anti-frode focalizzato su transazioni di pagamento (descrizione degli scenari d'uso, delle tipologie di transazioni, attori coinvolti, aspetti di sicurezza e scenari di attacco)
2	Definizione del problema di correlazione delle transazioni di pagamento e definizione dei requisiti
9	Analisi dello stato dell'arte relativo alla tecnologia di base di dati a grafo: identificazione delle diverse tecnologie disponibili, analisi degli elementi differenti, analisi dei requisiti e documentazione
5	Progettazione del prototipo dimostrativo sfruttando soluzione proposta
15	Implementazione della soluzione
3	Validazione e test
2	Documentazione
1	Presentazione interna dell'attività svolta

Tabella 2.1: Tabella della pianificazione del progetto di stage

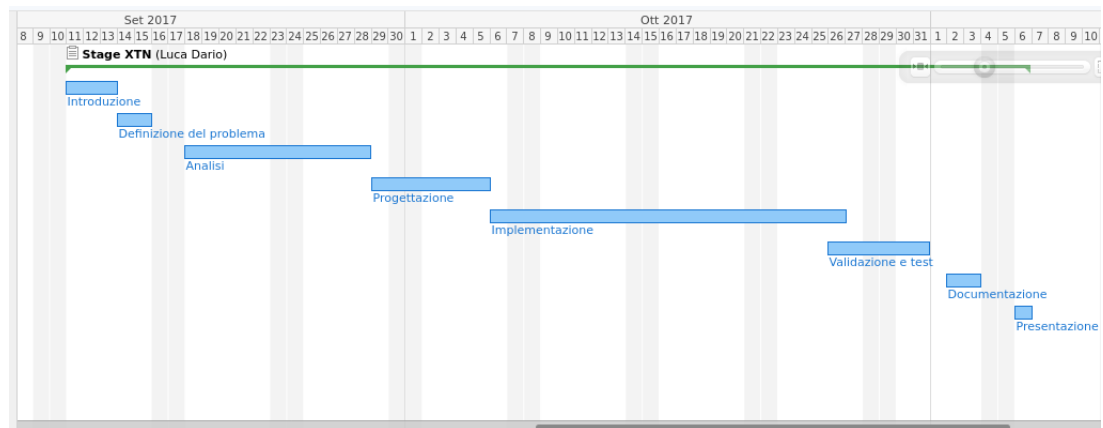


Figura 2.3: Diagramma di Gantt della pianificazione del progetto di stage

La pianificazione descritta nella tabella 2.1 soddisfa, quindi, i vincoli di durata attestandosi sui 40 giorni lavorativi corrispondenti a 320 ore.

2.1.5 Metodologia e interazione con il tutor aziendale

Ho deciso, in comune accordo con il tutor aziendale, che l'attività di stage dovesse essere svolta in loco presso la sede di Padova. Questo, oltre a favorire il dialogo con il tutor interno, dà l'opportunità di confrontarsi assieme ad un personale con esperienza decennale nel campo dello sviluppo ed anti frode.

2.2 Aspettative aziendali

Grazie al progetto di stage l'azienda spera di ricevere un'accurata analisi ed una possibile soluzione al problema affrontato precedentemente. Questo servirà in futuro come inizio di un **analisi di fattibilità** per verificare la possibilità di adottare le tecnologie affrontate anche nei loro prodotti. Un'analisi di questo genere in un'azienda rischierebbe di avere una priorità di esecuzione bassa con il rischio che questa attività non venga mai eseguita. Grazie al progetto di stage questo problema verrebbe soppiantato, insieme ad un grosso abbattimento dei costi.

2.3 Aspettative personali

I miei obiettivi iniziali prima della scelta del progetto di stage erano:

- * conoscere nuove tecnologie.
- * entrare a contatto con le dinamiche del mondo del lavoro.
- * poter offrire un contributo concreto ed utile all'azienda che mi ospitava.
- * condurre un buon stage per poter sviluppare una buona tesi.

Considerando i miei obiettivi e la lista delle aziende iscritte a *Stage-It*, ho deciso di scegliere questo progetto in particolare per la sua natura. Questo offre un buon compromesso tra ore spese per l'analisi ed ore per la progettazione e sviluppo. Le ore spese nell'analisi, oltre ad un **arricchimento personale**, mi permettono di scrivere una **buona tesi**. Invece le ore spese nello sviluppo mi permettono di **accrescere le mie conoscenze tecniche e di dinamiche aziendali**.

Successivamente alla mia scelta i miei obiettivi aumentarono, anche grazie al tema trattato dal progetto, tra cui:

- * Mettersi in gioco per cercare di concludere tutti gli obiettivi del piano di lavoro nel miglior modo possibile. Questo mi permetterebbe di eseguire una buona presentazione del lavoro svolto in azienda.
- * Conoscere le dinamiche di rilevazione di una frode bancaria.
- * Lavorare con persone con esperienza decennale.
- * Sviluppare conoscenze in un ambito nuovo, tra cui base di dati a grafo e *framework avanzati*.
- * Essere d'aiuto per risolvere un problema reale ad un'azienda significativa nel loro settore.

Capitolo 3

Resoconto dello stage

3.1 Pianificazione

Prima dell'inizio del progetto di stage, insieme al mio tutor aziendale, abbiamo stilato le attività principali che avrei dovuto svolgere durante il periodo a disposizione. Queste attività sono riportate nella tabella 2.1.

Il mio tutor ha deciso che, successivamente all'attività di analisi, avrei dovuto svolgere un colloquio con lui per decidere insieme le tecnologie più promettenti e valide nel contesto di anti frode.

Durante i primi giorni di stage abbiamo anche svolto vari colloqui per pianificare in dettaglio le varie attività, le modalità di scelta in base al loro ambito di sviluppo e le tecnologie consigliate.

3.2 Analisi

3.2.1 Introduzione

Nei primi giorni di stage io ed il mio tutor abbiamo deciso i principali punti su cui analizzare le varie tecnologie. Questi sono conseguenti all'ambito di sviluppo dell'azienda e le tecnologie che già adottano.

I punti di interesse sono:

- * **Licenza:** verificare che tipo di licenza ha il prodotto, se ha una versione gratuita e che tipo limitazioni ha rispetto a quella a pagamento.
- * **Tipo:** verificare il tipo di tecnologia implementata, se una base di dati a [grafo nativo](#) o [non nativa](#).
- * **Modelli disponibili:** verificare che tipi di [modelli](#) offre.
- * **Community:** verificare l'ampiezza della *community* del prodotto in analisi, avendola molto sviluppata è possibile ricercare consigli o risoluzioni di determinati problemi nei vari *forum* di riferimento.
- * **Tecnologie supportate nativamente (di rilievo per l'azienda):** verificare se la tecnologia mette a disposizione un supporto a tecnologie di riferimento all'azienda quali Java o Spring.

- * **Linguaggio di interrogazione proprietario:** verificare se la tecnologia in analisi mette a disposizione un linguaggio di interrogazione proprietario e ottimizzato per essa, analizzando la sua espressività.
- * **Supporto Tinkerpop¹:** verificare se la tecnologia mette a disposizione il supporto ad Apache Tinkerpop, sfruttando quindi un'interfaccia comune, permettendo poi il passaggio ad un'altra tecnologia che lo supporti a costi nulli.
- * **Clustering:** verificare se la tecnologia mette a disposizione un sistema di *clustering* pronto all'uso, il tipo e se ha la possibilità di eseguire *query* distribuite. Infine verificare se questa è disponibile nelle versioni gratuite.
- * **Security:** verificare che sistemi di sicurezza dei dati la tecnologia mette a disposizione e se esiste la possibilità di dividere il *database* in zone facendole diventare accessibili solo a determinati utenti.

Le informazioni necessarie all'analisi le ho reperite in primo luogo dalle documentazioni ufficiali delle case produttrici e successivamente dai *forum* quali StackOverflow² e Dzone³.

3.2.2 Tecnologie in analisi

Le tecnologie le ho scelte in base a quelle più diffuse o più promettenti sulla carta attraverso ricerche su internet.



Figura 3.1: Alternative tecnologiche nel mercato <https://goo.gl/d3UUgT>

Sparksee(DEX)

Sparksee⁴ è un grafo nativo sviluppato in C++ da Spark Technologies a fine 2008 sotto nome DEX. Successivamente nel 2014 cambia nome in Sparksee.

È stato il primo *database* a grafo disponibile per Android e iOS.

Supporta tecnologie di rilievo per l'azienda come Java, Maven e viene rilasciato per Linux, MacOS, Windows e per i principali sistemi operativi mobili. La casa produttrice

¹Tinkerpop: url= <http://tinkerpop.apache.org/>

²StackOverflow: url= <https://stackoverflow.com/>

³Dzone: url= <https://dzone.com/>

⁴Sparksee: url= <http://www.sparsity-technologies.com/>

dichiara come caso d'uso possibile il rilevamento di frode bancarie. Questa tecnologia viene rilasciata solamente in versione con licenza commerciale ed accademica con il limite ad un milione di nodi.

Neo4j

Neo4j⁵ è la base di dati a **grafo nativa** più diffusa al mondo, e, anche grazie a questo, ha una *community* molto ampia. È un *software* sviluppato in Java da Neo Technology nel 2007. È adottato da multinazionali come Microsoft, AirBnb, IBM, Ebay. Neo4j ha un linguaggio di interrogazione proprietario, chiamato Chyper, molto espressivo e ottimizzato. Supporta le tecnologie di rilievo per l'azienda, compreso Spring Data⁶. Neo4j viene rilasciato in versione *community* gratuita con nessuna limitazione di ampiezza della base di dati senza la possibilità di eseguirlo in modo distribuito, al contrario di quella *Enterprise* a pagamento.

ArangoDB

ArangoDB⁷ è un *database* multi modello sviluppato in C++ nato nel 2011. È un *database* principalmente documentale ma con collezioni che, raccogliendo la chiave di entrata e di uscita, simulano il funzionamento a grafo. Questo permette di organizzare i dati utilizzando il modello a grafo, documentale e chiave/valore.

Gli sviluppatori promettono la flessibilità data dalle base di dati multi modello e velocità paragonabile ai *database* a grafo nativi.

ArangoDB ha un linguaggio di interrogazione proprietario simile ad **SQL** e viene rilasciato in versione gratuita con limitazioni solo nell'ambito della sicurezza.

AllegroGraph

AllegroGraph⁸ è un software per *database* a grafo nativo sviluppato insieme agli *standard* W3C⁹ per il *web semantico* nel 2004. Supporta tecnologie di interesse all'azienda ma non ha un linguaggio di interrogazione proprietario. Viene rilasciato solo con licenza commerciale a pagamento.

OrientDB

OrientDB¹⁰ è un *software* per base di dati multi modello sviluppato in Java da uno sviluppatore italiano chiamato Luca Garulli¹¹. È un *database* principalmente documentale ma, salvando attraverso tabelle gli indici fisici di inizio e fine dell'arco, riesce ad implementare anche il modello a grafo nativo.

Per scelta puramente commerciale hanno deciso di integrare un linguaggio di interrogazione molto simile a **SQL**, limitando di molto l'espressività nelle ricerche nel modello a grafo. Viene rilasciato in versione sia a pagamento che *community* con nessuna limitazione di rilievo.

⁵Neo4j: url= <https://neo4j.com/>

⁶Spring data: url= <http://projects.spring.io/spring-data/>

⁷ArangoDB: url= <https://www.arangodb.com/>

⁸AllegroGraph: url= <https://franz.com/agraph/allegrograph/>

⁹W3C: url= <https://www.w3.org/>

¹⁰OrientDB: url= <http://orientdb.com/orientdb/>

¹¹Luca Garulli: url= <https://www.linkedin.com/in/garulli>

Titan

Titan¹² è un software in grado di sfruttare *storage backend* come Apache Cassandra¹³ per l'immagazzinamento dei dati. È capace di astrarre i dati nello *storage* ed organizzarli come se fossero a grafo, di conseguenza non utilizza una base di dati a *grafo nativa*. Questa possibilità di scelta dello *storage backend* lo rende molto valido se l'azienda utilizzasse una tecnologia che lo supporta, questo porterebbe ad un costo di passaggio tecnologico più contenuto.

Titan non offre un linguaggio di interrogazione proprietario ma supporta pienamente Tinkerpop. Viene rilasciato in licenza Apache 2¹⁴.

3.2.3 Scelta tecnologica



Figura 3.2: Logo *Neo4j* e *OrientDB*

Dopo un colloquio dove ho mostrato al mio tutor i risultati dell'analisi, in comune accordo abbiamo scelto le due tecnologie più promettenti ed interessanti.

Abbiamo scelto **Neo4j** per il suo linguaggio di interrogazione molto espressivo e la sua diffusione a livello mondiale che porta ad una *community* molto attiva.

Infine abbiamo scelto anche **OrientDB** essendo il *database* multi modello più promettente sulla carta che organizza i dati come uno a grafo nativo.

3.3 Sviluppo del prototipo

3.3.1 Analisi dei requisiti

La prima attività svolta, dopo l'analisi delle tecnologie, è stata l'analisi dei requisiti. Questo perché l'organizzazione della base di dati deve essere strutturata in modo tale da semplificare le operazioni che il prototipo andrà ad eseguire.

Ho tracciato tutti i requisiti utilizzando il seguente schema.

* **Importanza:** può assumere questi valori:

- **1:** indica un requisito obbligatorio;
- **2:** indica un requisito opzionale;

* **Tipo:** può assumere questi valori:

¹²Titan: url= <http://titan.thinkaurelius.com/>

¹³Apache Cassandra: url= <http://cassandra.apache.org/>

¹⁴Apache 2: url= <https://www.apache.org/licenses/LICENSE-2.0>

- **F:** indica un requisito funzionale;
- **Q:** indica un requisito di qualità;
- **V:** indica un requisito di vincolo.

* **Identificativo:** indica il codice identificativo del requisito, è un intero positivo crescente.

Ho definito tutti i requisiti tramite *brainstorming* con il mio tutor aziendale e definiti nella seguente tabella.

Requisito	Identificativo
Stesura documento con descrizione dettagliata dell'analisi svolta.	1V1
Predisposizione di un ambiente di lavoro con il <i>database</i> scelto utilizzando <i>deploy</i> tramite container docker.	1V2
Sviluppo prototipo	1V3
Preparazione ed esecuzione di una presentazione.	2V4
Estensione del <i>dataset</i> considerato e aggiunta di maggiore complessità nelle relazioni valutate.	2V5
Il prototipo deve permettere il calcolo della reputazione totale	1F6
Il prototipo deve permettere il calcolo della reputazione relativa	1F7
Il prototipo deve permettere l'aggiunta di una transazione	1F8
Il prototipo deve permettere l'aggiunta di un <i>AccountId</i>	1F9
Il prototipo deve permettere l'aggiunta di un <i>EntityId</i>	1F10
Il prototipo deve permettere di associare un <i>AccountId</i> ad un <i>EntityId</i>	1F11
Il prototipo deve permettere l'eliminazione di un <i>AccountId</i>	1F12
Il prototipo deve permettere l'eliminazione di un <i>EntityId</i>	1F13
La copertura dei test di unità deve essere maggiore o uguale al 85%	1Q14
Il prototipo deve permettere il calcolo dell'ammontare delle transazioni da una certa data in poi rispetto un <i>AccountId</i>	2F15
Il prototipo deve riuscire a calcolare la somma delle reputazioni totali di tutti gli <i>AccountId</i> associati ad un <i>EntityId</i>	2F16
Il prototipo deve riuscire a calcolare la somma delle reputazioni relative di tutti gli <i>AccountId</i> associati ad un <i>EntityId</i>	2F17
Soddisfacimento del 100% dei requisiti obbligatori	1F18

Tabella 3.1: Tabella tracciamento requisiti.

Tutti i requisiti descritti sono stati poi approvati dal mio tutor aziendale.

3.3.2 Tecnologie utilizzate

Tutte le tecnologie che ho utilizzato sono state accordate preventivamente con il tutor aziendale.

IntelliJ Idea Ultimate

Ho utilizzato IntelliJ Idea, con licenza accademica, come **IDE** in primo luogo per la conoscenza maturata dai progetti passati eseguiti in ambito universitario. Inoltre integra il supporto a JUnit¹⁵ per facilitare i *test*, a Maven per eseguire la *Build* direttamente dall'**IDE**, *debugger* avanzato. In aggiunta offre anche molte funzionalità utili come la possibilità di eseguire *refactoring* con la certezza che il comportamento resti invariato.

Java 8

Ho utilizzato Java come linguaggio di programmazione perché, in primo luogo, era un vincolo dettato dal piano di lavoro descritto nella [sezione 2.1.3](#). Inoltre questo linguaggio ha una sintassi molto regolare e semplice. Infine essendo uno dei linguaggi più diffusi al mondo ha una vasta quantità di librerie supportate.

Maven

Maven¹⁶ è uno strumento per la gestione di progetti *software* sviluppato da Apache Software Foundation¹⁷. Permette di dichiarare tutte le dipendenze e le varie versioni delle librerie in un unico file in formato XML¹⁸, separando le *directory* di progetto dalle librerie utilizzate. Maven scaricherà automaticamente le dipendenze e le nasconderà in cartelle separate dal progetto.

La *build standard* automatica consiste in diversi passi ordinati e dipendenti l'uno dall'altro tra cui la compilazione, l'esecuzione dei *test*, la verifica. Questo ciclo di *build standard* sarà completato solamente se tutte le varie fasi non genereranno errori.

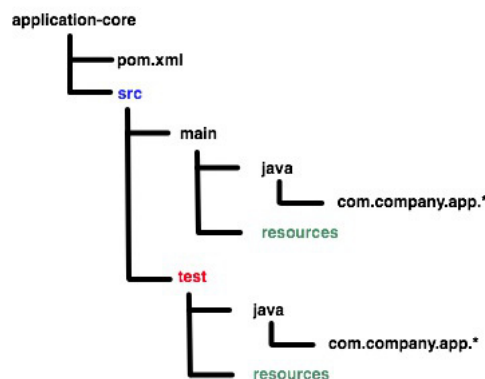


Figura 3.3: Struttura standard delle cartelle <https://goo.gl/N94Uqs>

¹⁵JUnit: url= <http://junit.org/junit5/>

¹⁶Maven: url= <https://maven.apache.org/>

¹⁷<https://www.apache.org/>

¹⁸XML: url<https://it.wikipedia.org/wiki/XML>

Spring Boot

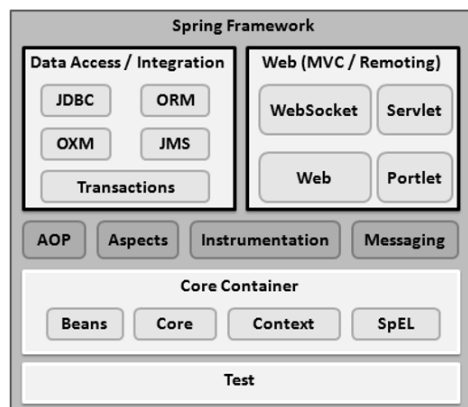


Figura 3.4: Architettura del *framework* Spring <https://goo.gl/2hvVYM>

Spring Boot¹⁹ è un *framework* per semplificare la creazione di progetti Java *stand-alone*. Con questo *framework* è possibile definire l'architettura interna di un progetto, ad esempio un *web server*, con semplici annotazioni Java. Per la creazione di un progetto Spring Boot basta recarsi in <https://start.spring.io/>, configurare il nome del progetto e scaricare il sorgente. Spring è una valida alternativa a Enterprise JavaBeans²⁰.

Spring Data Neo4j

Spring Data Neo4j²¹ è una particolare implementazione di Spring Data²². Questa libreria permette di astrarre le operazioni che si eseguono nella base di dati Neo4j semplificando lo sviluppo.

Utilizzando questa libreria è possibile definire delle *query* solo grazie alle segnatura dei metodi, questo permette sia velocizzare lo sviluppo sia l'attività di verifica visto che sarebbe inutile eseguirla in un metodo già verificato dalla casa produttrice. Infine permette di eseguire il *map* degli oggetti Java nel *database* semplicemente con delle annotazioni senza generazione di codice.

```
@NodeEntity(label = "AccountId")
public class AccountIdNeo4j implements BaseAccount {

    @GraphId
    private Long id;

    private String accountId;

    @Relationship(type = "TRANSACTION")
    private Vector<AccountIdNeo4j> transactions = new Vector<>();

    @Override
    public String getUniqueId() {
        return null;
    }
}
```

Figura 3.5: Esempio di Nodo utilizzando Spring Data Neo4j

¹⁹Spring Boot: url= <https://projects.spring.io/spring-boot/>

²⁰Enterprise JavaBeans: url= https://it.wikipedia.org/wiki/Enterprise_JavaBeans

²¹Spring Data Neo4j: url= <https://projects.spring.io/spring-data-neo4j/>

²²Spring Data: url= <http://projects.spring.io/spring-data/>

Tinkerpop

Ho utilizzato Tinkerpop²³ per interfacciarmi con OrientDB in modo semplice, diminuendo le operazioni eseguite esclusivamente tramite *query*. Questa libreria ha il vantaggio di permettere un passaggio ad un'altra tecnologia che lo supporta senza modificare il codice. E stata sviluppata da Apache Software Foundation con lo scopo di uniformare il codice volto a interfacciarsi con la maggior parte dei *database* nel mercato.

Graph Application Architecture

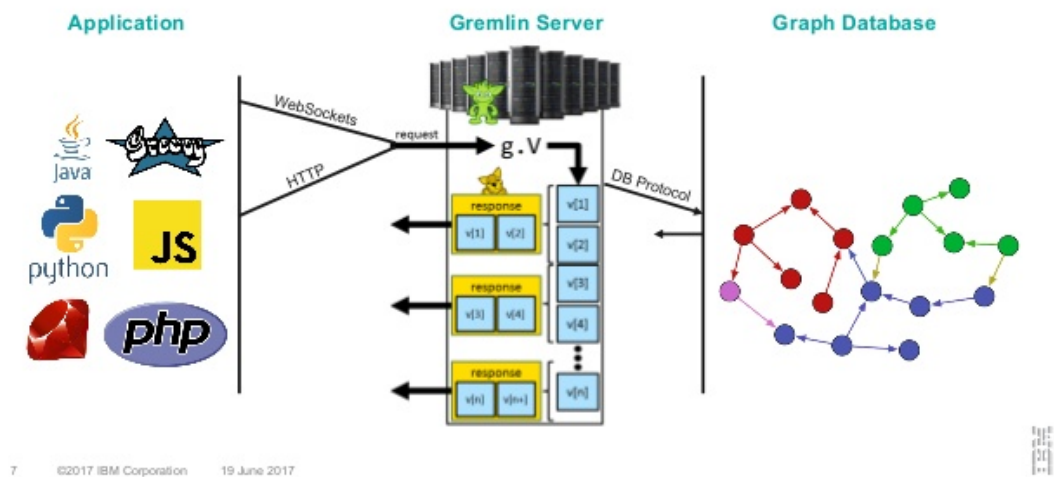


Figura 3.6: Architettura di un'applicazione che utilizza Tinkerpop <https://goo.gl/YiXuLB>

JUnit e Mockito

JUnit²⁴ è un **framework** che permette e semplifica la creazione di test in Java. Invece Mockito²⁵ permette di sovrascrivere il comportamento di un particolare metodo impostandogli i valori di entrata e di uscita. Questo è utile quando si sviluppano i *test* di unità e si utilizzano delle particolari classi non sviluppate o semplicemente non coperte da *test*.

²³Tinkerpop: url= <http://tinkerpop.apache.org/>

²⁴JUnit: url= <http://junit.org/junit5/>

²⁵Mockito: url= <http://site.mockito.org/>

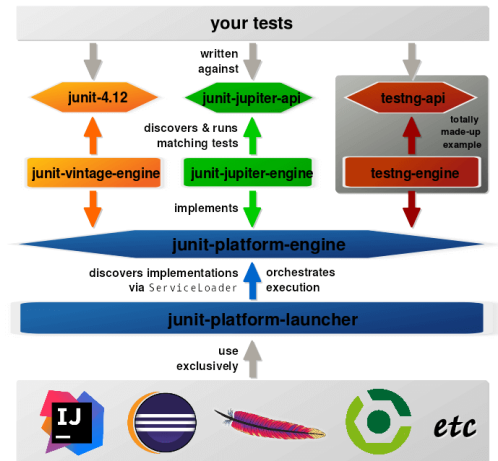


Figura 3.7: Architettura di JUnit <https://goo.gl/6Xgy5U>

3.3.3 Base di dati

In base al problema descritto nella sezione 2.1.1, dinamiche di Smash® ed i requisiti descritti nella tabella 3.1 ho organizzato i dati, per tutti e due i *database*, nel seguente modo:

* **Nodi:**

- **EntityId:** è il nodo che rappresenta un utente di una banca, esso può sia svolgere transazioni verso un *AccountId* che avere relazioni di appartenenza verso uno, o più, *AccountId*.
- **AccountId:** è il nodo che rappresenta una coordinata bancaria ad esempio un iban, esso può avere transazioni verso altri *AccountId* ed avere massimo una relazione di appartenenza da un *EntityId* a questo nodo.

* **Archi:**

- **TRANSACTION:** è l'arco che rappresenta una transazione, questo può provenire da un *AccountId* o *EntityId* verso un *AccountId*. Questo ha diverse proprietà tra cui di obbligatorie la data di avvenuta transazione, l'importo e la valuta, e di non obbligatorie come la città e il paese dove è stata eseguita la transazione.
- **OWN:** è l'arco che descrive la relazione di appartenenza da un *EntityId* ad un *AccountId*. Un *EntityId* può avere un numero a piacere di relazioni di appartenenza, al contrario un *AccountId* può avere al massimo un arco entrante di tipo *OWN*.

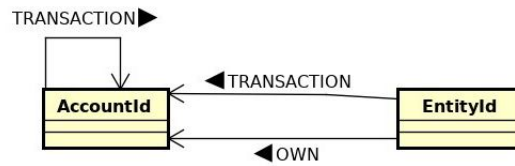


Figura 3.8: Schema della base di dati

A causa dei problemi legati alle politiche della protezione dei dati sensibili, non ho potuto utilizzare dati reali per il popolamento dei *database* ma ho dovuto creare un progetto per la generarmeli in modo casuale.

3.3.4 Prototipo

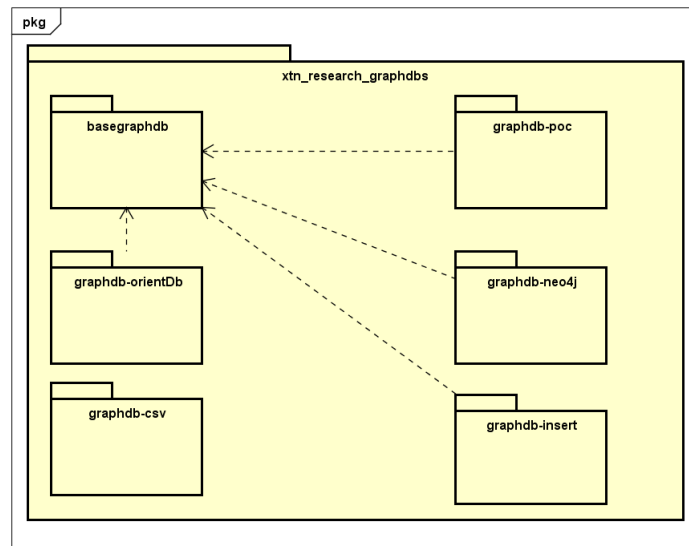


Figura 3.9: Diagramma dei package del prototipo

Ho deciso di dividere il prototipo in più progetti sotto l'unico [aggregato Maven](#) chiamato *xtn_research_graphdb*. Questa tecnica l'ho utilizzata per gestire tutte le versioni delle dipendenze in un unico file di configurazione, e, soprattutto, per avere la comodità di eseguire la *build* di tutti i progetti in un unico comando, lasciando a Maven l'onere di risolvere tutte le dipendenze.

basegraphdb

Basegraphdb è il progetto che contiene le varie interfacce comuni tra cui *ReputationController*. Questa è l'interfaccia comune per gestire la reputazione delle varie entità presenti nei vari *database*. Ogni implementazione, quindi sia *graphdb-neo4j* che *graphdb-orientDb*, conterrà una classe che la implementerà.

graphdb-orientdb

Graphdb-orientDb è il progetto per la gestione della reputazione tramite OrientDB, esso ha all'interno una classe chiamata *ReputationControllerOrientDb* che implementa *ReputationController*. Questa implementazione usa Tinkerpop²⁶ per interfacciarsi con il *database*.

graphdb-neo4j

Graphdb-neo4j è il progetto per la gestione della reputazione tramite Neo4j, esso ha all'interno una classe chiamata *ReputationControllerNeo4j* che implementa *ReputationController*. Questa implementazione utilizza *Spring Data Neo4j* per interfacciarsi ed eseguire il *map* degli oggetti Java nel *database*.

graphdb-csv

Graphdb-csv è il progetto per creare tutti i documenti in formato CSV contenenti le entità per popolare i diversi tipi di *database*. Questo crea dati in modo casuale nei migliori dei casi, cioè ogni *AccountId* ha il proprio *EntityId*.

graphdb-insert

Graphdb-insert inserisce in modo del tutto casuale un certo numero, definito dall'utente, di *EntityId*, *AccountId*, *Transaction* utilizzando l'interfaccia comune *ReputationController*.

Questo progetto deve essere usato solo per inserire un modesto numero di nodi ed archi visto che l'aggiunta di ogni *record* ci impiega 500 millisecondi. In caso contrario è consigliato generarsi i CSV delle entità utilizzando il progetto *graphdb-csv* e successivamente importarli nel *database* scelto.

graphdb-poc

Graphdb-poc è il progetto che si occupa solamente di istanziare le varie implementazioni di *ReputationController* ed eseguire i vari metodi d'interfaccia tracciando, e stampando a video, il nome del metodo, il risultato ottenuto ed il tempo impiegato.

Questo progetto non ha dipendenze verso le varie implementazioni ma solamente verso l'interfaccia. Questo è possibile delegando a Spring l'onere di iniettare tutte le classi che implementano l'interfaccia comune. In questo modo si ha un disaccoppiamento tra questo progetto e le varie implementazioni e, soprattutto, se in futuro ci fosse il bisogno di aggiungerne un'altra non sarebbe necessario modificare il codice di questo progetto.

²⁶Tinkerpop: url=<http://tinkerpop.apache.org/>

```

test per far istanziare i beans di Neo4j, essendo Lazy
-1 Neo4jReputazione Totale [ris: 523] millis: 1492
-1 OrientDBReputazione Totale [ris: 523] millis: 175

////////////////////////////////////
0 Neo4jReputazione Totale [ris: 523] millis: 49
0 OrientDBReputazione Totale [ris: 523] millis: 138

1 Neo4jReputazione relativa [ris: 1] millis: 67
1 OrientDBReputazione relativa [ris: 1] millis: 85

2 Neo4jReputazione globale [ris: 523] millis: 46
2 OrientDBReputazione globale [ris: 523] millis: 445

3 Neo4jReputazione globale relativa[ris: 1] millis: 827
3 OrientDBReputazione globale relativa[ris: 1] millis: 682

4 Neo4jTransazione nelle ultimi 15 giorni[ris: 73079.0] millis: 37
4 OrientDBTransazione nelle ultimi 15 giorni[ris: 73079.0] millis: 21

5 Neo4jtransazione senza gli ultimi 15 giorni[ris: 156423.0] millis: 29
5 OrientDBtransazione senza gli ultimi 15 giorni[ris: 156423.0] millis: 29

```

Figura 3.10: Risultati stampati da graphdb-poc

3.4 Utilizzo della libreria

3.4.1 Calcolo della reputazione

Sfruttando la *dependency injection* ²⁷ di Spring, per utilizzare la libreria del calcolo della reputazione è sufficiente creare una classe simile alla seguente.

```

package it.ldario;

import it.ldario.graphdbneo4j.ReputationControllerNeo4j;
//import it.ldario.graphdborientDb.ReputationControllerOrientDb;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MiaClasse implements CommandLineRunner {

    public static void main(String[] args) {
        SpringApplication.run(MiaClasse.class, args);
    }

    //utilizzando, come tipo statico, ReputationControllerNeo4j oppure
    //ReputationControllerOrientDb verra' usato, rispettivamente
    //l'implementazione di Neo4j o OrientDB

```

²⁷dependency injection: url= https://en.wikipedia.org/wiki/Dependency_injection

```

private ReputationControllerNeo4j reputationController;
//private ReputationControllerOrientDb reputationController;

@Autowired
public MiaClasse(ReputationControllerNeo4j reputationController) {
    this.reputationController = reputationController;
}

@Override
public void run(String... strings) throws Exception {
    //aggiunge al database scelto(orientDB oppure Neo4j), dipende dal tipo
    //che viene dichiarato, un entity con id="Luca"
    reputationController.addAccountId("Luca");

    //stampa la reputazione totale dell'entity "Luca"
    System.out.println(reputationController.getTotalReputation("Luca"));

    //stampa il totale delle transazioni effettuate da Luca negli ultimi
    //15 giorni
    System.out.println(reputationController
        .getAmountAccountIdInATimeRange("Luca", 15));

    //aggiunge al database scelto(orientDB oppure Neo4j), dipende dal tipo
    //che viene dichiarato, un account con id "IT123"

    TransactionBuilder transactionBuilder = new TransactionBuilder();
    transactionBuilder.setCountry("Italy").setCity("San Michele delle
        Badesse").setAmount(100).setCurrency("Euro");
    reputationController.addTransaction("Luca", "IT123", transactionBuilder
        .build());
}
}

```

3.4.2 Estensione della libreria con un altro tipo di database

Per aggiungere un altro tipo di database è necessario solamente creare una classe simile alla seguente, senza modificare il codice del prototipo.

```

package it.ldario.newdatabase;

import it.ldario.basegraphdb.ReputationController;

public class ReputationControllerNewDatabase implements ReputationController {

```

```
//implementazione di tutti i metodi di interfaccia  
}
```

3.4.3 Popolamento della base di dati

Per aggiungere un numero modesto di dati generati casualmente, in qualunque tecnologia che implementa *ReputationController*, è necessario solamente creare una classe simile alla seguente.

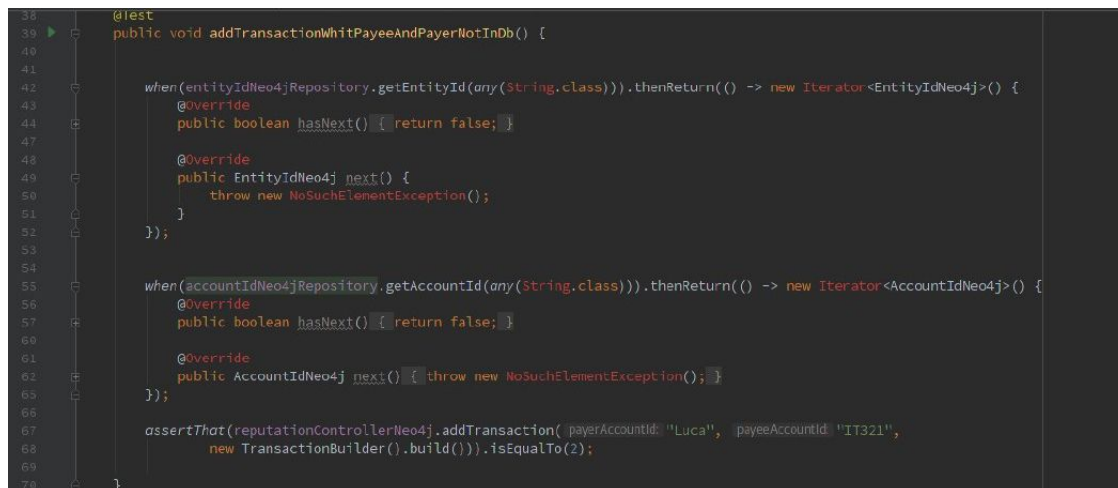
```
package it.ldario;  
  
import it.ldario.graphdbinsert.GraphDbInsertImpl;  
import it.ldario.graphdbneo4j.ReputationControllerNeo4j;  
//import it.ldario.graphdborientDb.ReputationControllerOrientDb;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
  
public class MiaClasse implements CommandLineRunner {  
  
    //esplicitando il tipo verra' scelto, rispettivamente, il reputation  
    //controller di Neo4j o OrientDB  
    private ReputationControllerNeo4j reputationController;  
    //private ReputationControllerOrientDb reputationController;  
  
    public static void main(String[] args) {  
        SpringApplication.run(GraphdbPocApplication.class, args);  
    }  
  
    @Autowired  
    public MiaClasse(ReputationControllerNeo4j reputationController) {  
        this.reputationController = reputationController;  
    }  
  
    @Override  
    public void run(String... strings) throws Exception {  
        GraphDbInsertImpl graphDbInsert = new  
            GraphDbInsertImpl(10,20,50,reputationController);  
  
    }  
}
```

Eseguendo il *Main* si andranno ad aggiungere nel *database* scelto 10 *EntityId*, 20 *AccountId* e 50 Transazioni. Per scegliere la base di dati su cui agire è necessario modificare, con il tipo desiderato, la variabile *reputationController*.

3.5 Verifica e validazione

3.5.1 Verifica

Nell'attività di verifica ho utilizzato JUnit²⁸ e Mockito²⁹ per facilitare la creazione dei *test*.



```

38  @test
39  public void addTransactionWhitPayeeAndPayerNotInDb() {
40
41
42      when(entityIdNeo4jRepository.getEntityId(any(String.class))).thenReturn(() -> new Iterator<EntityIdNeo4j>() {
43          @Override
44          public boolean hasNext() { return false; }
45
46          @Override
47          public EntityIdNeo4j next() {
48              throw new NoSuchElementException();
49          }
50      });
51
52
53
54
55      when(accountIdNeo4jRepository.getAccountId(any(String.class))).thenReturn(() -> new Iterator<AccountIdNeo4j>() {
56          @Override
57          public boolean hasNext() { return false; }
58
59          @Override
60          public AccountIdNeo4j next() { throw new NoSuchElementException(); }
61      });
62
63
64
65
66      assertThat(reputationControllerNeo4j.addTransaction( payerAccountId: "Luca", payeeAccountId: "IT321",
67          new TransactionBuilder().build()).isEqualTo(2);
68
69
70  }

```

Figura 3.11: Esempio di test utilizzando Mockito e JUnit

Sono state testate tutte le *query* utilizzando un immagine docker³⁰ di *test*, escluse quelle banali auto generate da Spring. Invece per testare le varie unità mi sono aiutato istanziando ed istruendo facilmente dei *mock* utilizzando Mockito. In accordo con il mio tutor aziendale è stata decisa la soglia minima di copertura dei test del 85%. La copertura finale dei test si è attestata sul 88% tra il codice consegnato in azienda. Ogni test ha uno ed uno solo confronto al suo interno ed ha un nome che indica con precisione che operazioni andrà ad eseguire. Questo ha il vantaggio di sapere esattamente con che condizioni il *test* fallisce solamente leggendo il nome. Inoltre ogni requisito è stato coperto da *test*, questo per verificare l'avvenuto soddisfacimento di esso.

Tutti i *test* verificano la logica interna dell'unità e percorrono la quasi totalità dei cammini possibili, questo per far emergere tutti i possibili errori nel codice.

²⁸JUnit: url= <http://junit.org/junit5/>

²⁹Mockito: url= <http://site.mockito.org/>

³⁰Docker: url= <https://www.docker.com/>

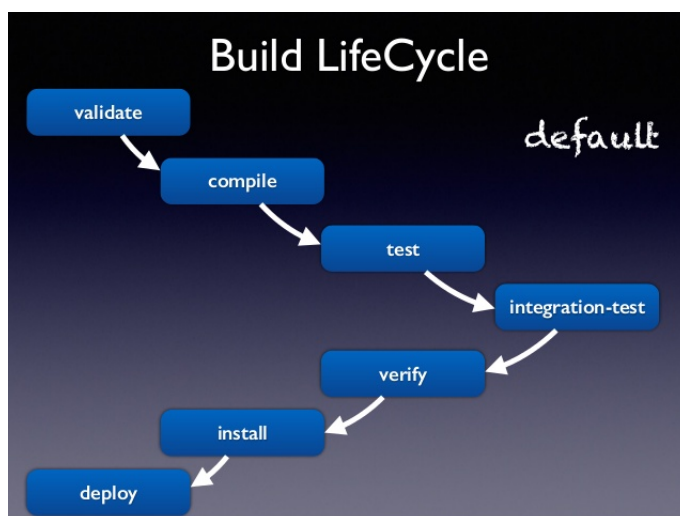


Figura 3.12: Ciclo di vita della build standard di Maven <https://goo.gl/gvgZBN>

La *build* non andrà a buon fine se tutti i *test* non avranno esito positivo, questo è gestito completamente e garantito dal processo di *build standard* di Maven.

3.5.2 Validazione

Per quanto riguarda la validazione ho consegnato un documento, in azienda, contenente una tabella con il codice del requisito descritto nella [tabella 3.1](#) e se è stato soddisfatto. La tabella dei requisiti obbligatori è la seguente:

Requisito	S ³¹ /NS ³²
1V1	S
1V2	S
1V3	S
1F6	S
1F7	S
1F8	S
1F9	S
1F10	S
1F11	S
1F12	S
1F13	S
1Q14	S
1F18	S

Tabella 3.2: Tabella soddisfacimento requisiti obbligatori

Invece la tabella dei requisiti opzionali è la seguente:

Requisito	S ³³ /NS ³⁴
2V4	S
2V5	NS
2F15	S
2F16	S
2F17	S

Tabella 3.3: Tabella soddisfacimento requisiti opzionali

Il requisito segnato con l'identificativo 2V5, cioè l'estensione del *dataset*, non è stato soddisfatto per mancanza di tempo. Ho preferito consumare il tempo rimanente per eseguire una buona presentazione interna all'azienda. Ho soddisfatto il 100% dei requisiti obbligatori e l'80% di quelli opzionali.

3.6 Conclusioni

3.6.1 Casi d'uso di una base di dati a grafo

Durante la mia analisi mi sono reso conto di come una base di dati a grafo non possa essere utilizzata per immagazzinare qualunque tipo di dato, anzi gli usi sono molto ristretti. Un *database* di questa tipologia è rivolto a chi deve salvare un altissimo numero di dati legato ad una cerchia di entità, di numero minore rispetto a questi. Solitamente questo altissimo numero di dati rappresentano gli archi e le entità rappresentano i nodi. Questo succede perché le potenzialità delle basi di dati di questo tipo emergono quando si ha un alto numero di archi collegati ad un modesto numero di nodi. Se avesse l'esatto opposto, quindi molti vertici e pochi archi, le operazioni disponibili sull'attraversamento tra i nodi si ridurrebbero di molto. Inoltre una base di dati di questo tipo non è adatta per operazioni che coinvolgono tutto il *database*.³⁵

³⁵The good and the bad about graph databases : url= <https://goo.gl/bRMwnv>

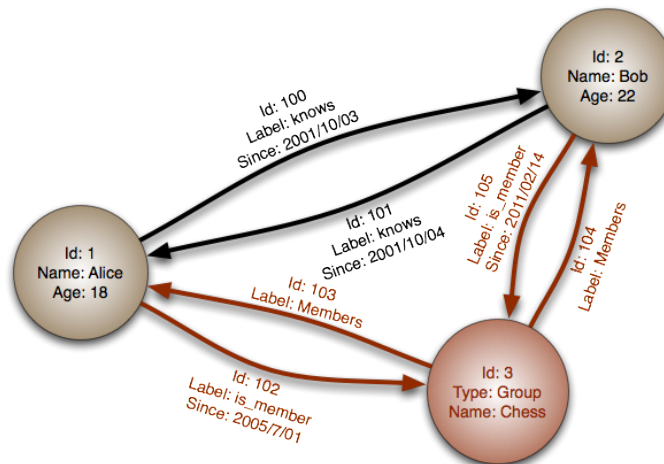


Figura 3.13: Base di dati per un social network <https://goo.gl/Fik4Xp>

Un esempio classico di ottimo caso d'uso è quello descritto dalla [figura 3.8](#), ossia la modellazione della base di dati di un social network. In questo caso il numero dei nodi, cioè le persone fisiche, sono in numero nettamente minore rispetto alla somma di tutti gli archi, cioè le conoscenze. Inoltre le operazioni che si eseguono coinvolgono sempre una base di dati circoscritta.

3.6.2 Vantaggi e svantaggi di una base di dati a grafo

Durante la mia analisi ho riscontrato i seguenti vantaggi e svantaggi nell'utilizzare una base di dati a grafo.

Vantaggi

- * **Velocità:** se il caso d'uso è adatto e si organizzano i dati in modo intelligente si riesce a ridurre la maggior parte delle *query* ad operazioni banali, sfruttando la natura dei grafi. Nell'ambito anti frode molte operazioni si riducono al "*conta quante volte la persona A ha fatto una determinata cosa, chiamata Z, verso un'entità B*" dove A e B sono nodi e Z è l'arco che li collega.
- * **Espressività della ricerca:** se il caso d'uso che si va a modellare è adatto alla natura dei grafi, l'espressività della ricerca è tale che le *query* risultano, molte volte, banali anche dal punto di vista della scrittura.
- * **Tutto è collegato:** data dalla natura dei grafi, si ha un'enorme possibilità di creare ricerche che esplorano in profondità il grafo date determinate condizioni, ad esempio la possibilità di attraversare e memorizzare solo i nodi che sono collegati da una determinata entità solamente tramite relazioni di appartenenza. Nell'ambito anti frode succede molto spesso di eseguire delle ricerche che sfruttano questa caratteristica.

Svantaggi

- * **Casi d'uso limitati:** solo pochi *use case* sono adatti ad essere modellati come un grafo, quindi è molto rischioso modellare totalmente la propria base di dati utilizzando questa tecnologia. Questo perché in futuro potrebbe esserci la necessità di aggiungere dati al modello non adatti a questo tipo di *database*. Questo limita il normale ciclo di vita di un *software*.
- * **Tecnologia di nicchia:** essendo una tecnologia di nicchia la possibilità di informarsi, tramite community, è ridotta rispetto ad altre tecnologie più diffuse, come possono essere i *database* relazionali.

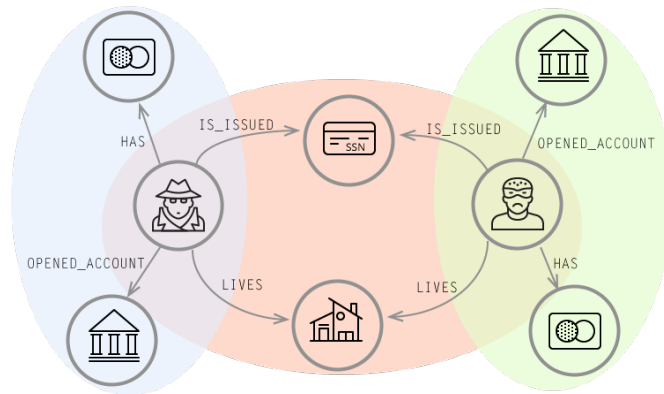


Figura 3.14: Esempio di un caso d'uso orientato all'anti frode <http://archive.is/LpXtZ>

3.6.3 PRO e CONTRO Neo4j rispetto OrientDB

PRO

- * **Espressività:** chyper, il linguaggio di interrogazione di Neo4j, inizialmente può spaventare, essendo diverso da quelli che siamo abituati a vedere. Dopo un breve periodo di apprendimento risulta, però, facile da utilizzare e molto potente nelle ricerche. Questo è dato dalla sintassi molto regolare ed espressiva con pochissime nozioni.
- * **Spring Data Neo4j:** libreria ben fatta e documentata per astrarre l'interazione con la base di dati, al contrario di OrientDB.
- * **Importatore da CSV:** popolamento della base di dati tramite documento CSV nettamente più veloce rispetto ad OrientDB.

CONTRO

- * **Input/Output lento:** le librerie di Neo4j, quali Spring Data ed in generale i *driver* java, risultano più lenti rispetto ad quelli di OrientDB. Questo l'ho notato calcolando la differenza tra la velocità netta delle *query*, eseguendola nella *dashboard*, e quella tramite i *driver*, successivamente confrontandola con OrientDB.

3.6.4 PRO e CONTRO OrientDB rispetto Neo4j

PRO

- * **Velocità:** pur essendo un *database* multi modello la velocità nelle ricerche è comparabile ad una base di dati a grafo.

CONTRO

- * **Spring Data OrientDB:** non documentato ed acerbo. Questo mi ha portato ad usare Tinkerpop³⁶ portando allo sviluppo di un software meno organizzato rispetto a quello di Neo4j.
- * **Espressività:** l'azienda produttrice di OrientDB ha fatto la scelta, per me puramente commerciale, di utilizzare un linguaggio simile ad SQL per l'interrogazione della base di dati. Questo diminuisce sostanzialmente l'espressività delle *query*, rendendo complicato lo sviluppo di operazioni con svariate transazioni su archi.
- * **Spazio su disco:** per rendere un database multi modello nativo gli sviluppatori di OrientDB hanno dovuto replicare molti dati, aumentando di circa 0.4 volte lo spazio necessario per immagazzinare i dati (prova eseguita con 30 milioni di record tra nodi ed archi).

3.6.5 Conclusioni

Nella parte finale del progetto di stage mi sono posto le seguenti domande per cercare di effettuare una presentazione esaustiva, in azienda, dell'analisi effettuata.

Vale la pena adottare la base di dati a grafo in azienda?

Secondo la mia analisi sì, però solamente per alcuni casi d'uso.

Adottare un *database* a grafo, e modellare i dati di conseguenza, per determinati casi d'uso renderebbero possibili determinate ricerche che prima potevano risultare troppo onerose o, addirittura, impossibili.

Il tratto caratteristico dei *database* a grafo è la possibilità di attraversare, a costo costante, le varie entità ricostruendo le azioni compiute da una particolare entità indipendentemente dalla grandezza della base di dati. Questo aprirebbe, per l'azienda, nuove possibili ricerche come la costruzione dell'albero delle transazioni, dato un utente, per estrapolare determinate informazioni.

Risolve il problema iniziale del calcolo della reputazione descritto nella sezione 2.1.1?

Sì, almeno utilizzando la base di dati generata casualmente.

Ho eseguito le operazioni in basi di dati con varie unità di grandezza, fino ad arrivare a 30 milioni di transazioni, e il tempo necessario per calcolare la reputazione si aggirava tra i 2 millisecondi per quella totale e i 15 millisecondi per quella relativa. Questo grazie alla possibilità di eseguire queste determinate operazioni senza caricare tutto il *database* in memoria.

Quale tecnologia utilizzare tra Neo4j e OrientDB?

Secondo la mia analisi Neo4j.

Ho preso questa decisione analizzandoli nei seguenti 3 ambiti, in ordine di importanza:

- * **Espressività:** il linguaggio di interrogazione di Neo4j è nettamente migliore in termini di espressività. SQL è chiaramente un linguaggio non adatto per le ricerche nei grafi. Al contrario, quello di Neo4j, è sempre espressivo sia in termini di lettura di una *query* sia nel crearla. Questo rende meno costoso, in termini di studio, il cambio tecnologico in azienda.

³⁶Tinkerpop: url= <http://tinkerpop.apache.org/>

- * **Supporto alle tecnologie usate in azienda:** il passaggio a Neo4j risulterebbe meno costoso, in termini di studio individuale, grazie al supporto nativo verso Spring Data. Il supporto di OrientDB verso questa tecnologia risulta ancora acerbo.
- * **Velocità:** in termini di velocità di esecuzione di una *query* Neo4j risulta migliore, soprattutto all'aumentare delle transazioni sui nodi.

Capitolo 4

Valutazione

4.1 Soddisfacimento degli obiettivi

Il progetto di stage ha avuto una durata di 316 ore, oltre la durata minima ma in linea con quella massima di 320 ore.

Negli ultimi due giorni di stage ho preparato, e successivamente esposto, una presentazione del lavoro svolto, durante il periodo di stage, al mio tutor, Giuseppe Pavan, al *team leader* e sviluppatore *senior*, Riccardo Cardin, e al CTO di *XTN – Cognitive Security*, Guido Ronchetti ¹. Grazie a questa presentazione ho potuto ricevere un *feedback* del lavoro svolto, riassumere al *team* dell'azienda che mi ha ospitato la mia analisi tecnologica e le idee su questi tipi di base di dati ed, infine, fare esperienza nella preparazione di un discorso tecnico da esporre a persone con esperienza decennale.

Attraverso i documenti consegnati, il prototipo e la presentazione da me svolti, l'azienda ha potuto acquisire le informazioni necessarie per valutare il passaggio a questo tipo di tecnologia. Chiaramente questo mio progetto non può sostituire una vera analisi di fattibilità con i relativi costi per lo studio di nuove tecnologie e conseguenze tecniche dovute al cambiamento. Il mio prodotto potrà servire, però, all'azienda per determinare se vale la pena svolgere questo tipo di attività e individuare nuove funzionalità dovute al cambiamento. Tutto questo risparmiando sia risorse sia tempo, che a volte in un'azienda possono essere limitate.

Gli obiettivi obbligatori, come descritto nella [sezione 2.1.3](#), si concentravano sull'analisi delle varie tecnologie di basi di dati a grafo e sulla progettazione e realizzazione di un prototipo che esemplifichi le capacità di queste. Sono stati soddisfatti completamente risparmiando un numero significativo di ore lavoro. Con le ore rimaste mi sono concentrato nel migliorare il prototipo, aggiungendo funzionalità interessanti, e nel soddisfare un requisito opzionale, cioè la preparazione di una presentazione interna del lavoro svolto.

¹Guido Ronchetti: LinkedIn= <https://www.linkedin.com/in/guido-ronchetti-285637a/>

4.2 Conoscenze acquisite

Le conoscenze che, grazie a questa esperienza di stage, ho acquisito o semplicemente ampliato sono le seguenti.

Base di dati a grafo

Durante questo progetto ho potuto avvicinarmi ai *database* organizzati a grafo, tecnologia che ignoravo prima di questa esperienza.

Inizialmente, questa tecnologia mi è sembrata simile alle basi di dati relazionali. Informandomi e testandola con il mio prototipo, però, ho compreso le sue potenzialità. Quasi tutti i vantaggi sono dati dalla possibilità di attraversare i nodi in tempo costante e non dipendente dalla grandezza del *database*, questo non avviene in quelli relazionali a causa delle operazioni di **JOIN**.

Mi ritengo soddisfatto di aver acquisito delle conoscenze in questa materia grazie all'evoluzione che sta avendo in questo ultimo periodo, rispetto alle altre tipologie di base di dati.

Popularity changes per category, October 2015

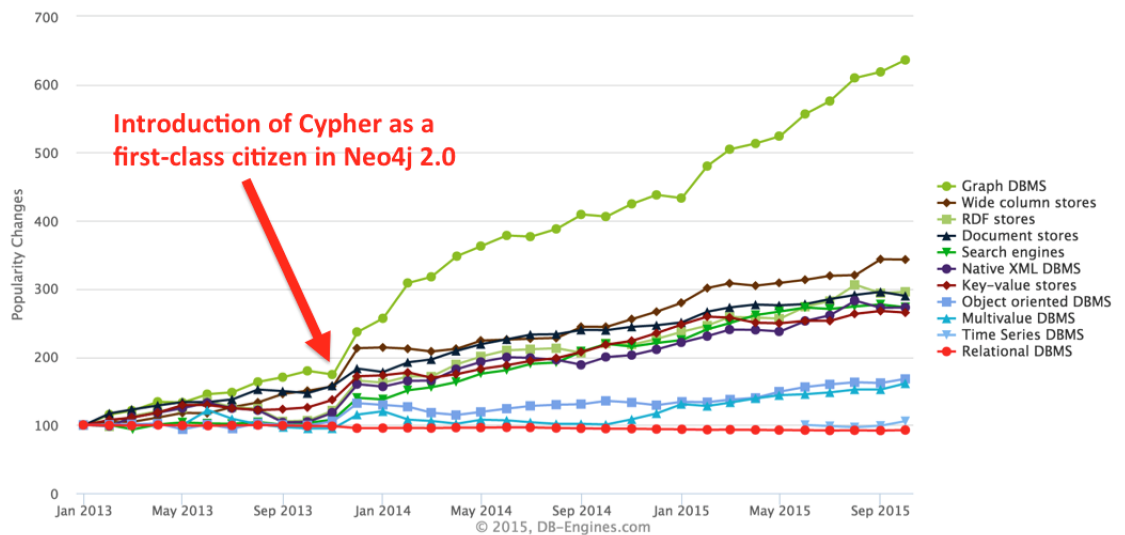


Figura 4.1: Evoluzioni delle base di dati a grafo rispetto alle altre tipologie
<https://goo.gl/SfV3vZ>

Capacità di condurre una analisi tecnologica

Attraverso questa esperienza ho potuto acquisire capacità di condurre una analisi tecnologica strutturata. Ho imparato ad individuare le possibili alternative, eseguire una scrematura utilizzando un confronto strutturato ed uniforme per caratteristiche ed, infine, scegliere la migliore in base al caso d'uso di riferimento. Tutto questo aiutandomi con le documentazioni ufficiali delle alternative tecnologiche e *forum* di riferimento quali StackOverflow² e Dzone³.

Progettazione e realizzazione di un'applicazione Java

Ho potuto acquisire capacità di progettare e realizzare autonomamente una applicazione Java sfruttando *framework* avanzati come Spring⁴ e utilizzare, anche se superficialmente, la tecnologia Docker⁵.

4.3 Valutazione personale

Per quanto riguarda questa esperienza di stage mi ritengo soddisfatto del lavoro svolto, in termini di requisiti e qualità del prodotto finale, e del progetto di stage scelto. Purtroppo non ho potuto soddisfare completamente i requisiti opzionali sia per mancanza di tempo che per l'impossibilità di recuperare, o solamente analizzare, dati reali per le stringenti politiche sulla *privacy* di dati sensibili.

Mi ritengo soddisfatto, invece, del progetto scelto perchè lo ritengo un giusto compromesso tra una analisi, che mi ha portato a svolgere una tesi con buoni contenuti, ed un progetto di sviluppo che mi ha permesso di acquisire conoscenze che sfrutterò nel mondo del lavoro.

Infine ritengo il progetto di stage, in un'azienda esterna, l'unico modo per acquisire nozioni che durante il corso di studi sarebbe molto difficile conseguire in così poco tempo. Personalmente estenderei l'obbligo di stage a molti altri corsi di studi universitari che non lo contengono.

4.4 Differenza tra università e mondo del lavoro

Durante la mia esperienza lavorativa ho notato un *gap* rispetto al percorso di studi universitario. Questa differenza si focalizza nella mancanza di conoscenza nelle tecnologie avanzate, come particolari *framework*, oppure di nuova generazione. Questa mancanza è sopperita, però, dall'insegnamento durante i corsi universitari dei concetti di base che mi hanno portato ad imparare ed adattarmi velocemente a queste nuove tecnologie. Ciò non significa che non si debba aggiornare i percorsi di studi con un programma adatto al mondo lavorativo odierno.

Personalmente valuterei l'aggiunta, da parte dell'università, di un insegnamento che avvicini lo studente alle basi di dati non relazionali e l'uso avanzato di Java, spiegando le nuove funzionalità introdotte con le recenti versioni di questo linguaggio.

Inoltre aggiungerei, ad un corso già presente, un avvicinamento ai sistemi di controllo di versione, invece di delegare totalmente allo studente l'onere di impararlo durante gli svariati progetti universitari.

²StackOverflow: url= <https://stackoverflow.com/>

³Dzone: url= <https://dzone.com/>

⁴Spring: url= <https://spring.io/>

⁵Docker: url= <https://www.docker.com/>

Un'altra cosa che mi ha lasciato perplesso durante il mio percorso universitario è stata la scelta di non insegnare, nel corso di tecnologie *web*, tecnologie di rilievo, anche se non definite da *standard*, come HTML5⁶.

Al netto di questi dettagli il mio percorso di studi mi ha permesso di acquisire nozioni ed aspetti fondamentali dell'informatica che non sarei riuscito ad imparare autonomamente. Inoltre è riuscito a far sì che mi appassionassi alla materia nonostante, prima dell'iscrizione all'Università di Padova, la mia conoscenza a questa fosse nulla.

Considero molto importante la presenza dei progetti universitari: mi hanno insegnato a lavorare in un *team*, mantenuto alta la voglia di migliorarmi e permesso di aggiungere un'esperienza dimostrabile al mio *curriculum*, oltre a tutte le nozioni tecniche che ho appreso.

⁶HTML5: url= https://www.w3schools.com/html/html5_intro.asp

Bibliografia

4.5 Siti Consultati

Neo4j: <https://neo4j.com/>

OrientDB: <http://orientdb.com/orientdb/>

ArangoDB: <https://www.arangodb.com/>

AllegroGraph: <https://franz.com/agraph/allegrograph/>

TitanDB: <http://titan.thinkaurelius.com/>

DZone: <https://dzone.com/>

StackOverflow: <https://stackoverflow.com/>

Spring: <https://spring.io/>

Docker: <https://www.docker.com/>

Apache Tinkerpop: <http://tinkerpop.apache.org/>