



Dipartimento di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica

Tesi di Laurea

**Realizzazione di un data provider per  
l'integrazione e l'esportazione di dati finanziari**

Laureando

Antonio Martinelli

**Relatore:**

Prof. Luca Cabibbo

**Correlatore:**

Ing. Luigi Bellomarini

Anno Accademico 2014/2015



# Ringraziamenti

Ci tengo a ringraziare anzitutto il professor Luca Cabibbo, Relatore, e l'ingegnere Luigi Bellomarini, Correlatore, per i preziosi consigli con i quali ho migliorato ed arricchito il contenuto di questa tesi e per la possibilità di svolgere questa esperienza formativa.

Proseguo ringraziando i miei colleghi e amici con i quali ho affrontato questo ciclo di studi e grazie ai quali sono qui ora, tra cui Simone Rocchi, Claudia Romeo, Antonio Matinata, Lorenzo Goldoni ed Alessio Zoccoli, ed in particolare Marco Faretra, con il quale ho collaborato in team durante il tirocinio, e Federico Ginosa, pieno di preziosi consigli. Ringrazio inoltre il mio amico Luca, che mi sopporta quasi da una vita.

Inoltre, ringrazio tutti i miei parenti, in particolare le mie zie Anna e Patrizia “Parsidia”, che mi hanno sempre sostenuto e fornito l’“hardware” dal quale scrivo, con il quale ho semplificato e velocizzato enormemente questo percorso, e mio cugino Marco, che mi ha sopportato fin da piccolo e mi ha appassionato all’informatica. Molto devo a lui per la scelta di questo corso di laurea, oltre ad utili suggerimenti durante il corso degli studi.

Infine, un ringraziamento speciale va ai miei genitori, che ringrazio per il loro grande amore e per le infinite possibilità che mi hanno sempre offerto, a mia sorella Claudia, futura “Ingegnera”, e ad Elena, a cui dedico questo lavoro, e della quale spero di leggere entro pochi mesi una tesi. Grazie a voi, che mi sostenete da tanto e mi spronate ad andare avanti.

# Introduzione

L'obiettivo del tirocinio è stato l'ampliamento di un'applicazione web, Glimpse, per la raccolta, l'elaborazione e la condivisione di serie storiche tramite l'analisi, la progettazione e lo sviluppo di nuove funzionalità.

In questi ambiti, molte serie storiche sono raccolte, mantenute e organizzate da banche centrali, banche nazionali, istituti statistici nazionali e commercial data providers nei rispettivi databases. Tuttavia la fruibilità di questi dati è limitata: ciascuna sorgente li rappresenta in un formato diverso e l'utilizzatore finale deve relazionarsi con molte sorgenti eterogenee, non avendone a disposizione una integrata. Infatti, sebbene per la rappresentazione di questo genere di dati esistano molti standard, questi sono raramente adottati o, in ogni caso, non in modo completo.

In questo senso, l'integrazione di varie sorgenti, rappresenta un problema importante per il ricercatore, lo statistico e l'economista. In effetti, l'onere di dell'integrazione aumenta il tempo necessario per il raggiungimento dell'obiettivo di business, quale ad esempio un'analisi economica o la produzione di un articolo scientifico.

Glimpse risolve questo problema. Si propone come intermediario tra gli economisti e i dati loro necessari, mediante la raccolta di questi ultimi attraverso interfacce con i vari database delle fonti, quali ad esempio FRED, WORLDBANK, OECD, IMF ed ECB. I dati così raccolti vengono salvati, integrati e frequentemente aggiornati su un database interno, man mano che gli utenti li richiedono. Il caricamento non viene quindi effettuato a priori, ma utilizza un approccio "*lazy*", basato sulle richieste ricevute.

Gli interfacciamenti possono richiedere tecniche più o meno complesse poiché non c'è uniformità tra i formati con cui le fonti conservano le serie.

La grande intuizione di Glimpse è la conversione di questi dati in un unico formato, con la creazione di una banca dati interna che si incrementa con l'utilizzo del software da parte dell'utenza.

Viene inoltre offerta la possibilità di eseguire trasformazioni matematiche, utili per evidenziare la tendenza di fondo o ridurre la variabilità e per evidenziare altri fattori utili durante lo studio delle serie storiche.

Infine, è possibile esportare il dato in vari formati, come fogli di calcolo, immagini e file in linguaggi di programmazione familiari al bacino d'utenza, per svolgere analisi, oppure in Latex per l'integrazione in trattati.

I formati sono in numero sempre maggiore, data la facilità di traduzione partendo da una singola struttura.

Uno dei principali valori di Glimpse è quello di fornire trasparenza ed un'interfaccia molto semplice all'utente, infatti quest'ultimo può inserire delle parole significative per trovare la time series a cui è interessato, e poi scorrere i risultati trovati per scegliere quello giusto, oppure, conoscendo l'identificatore della serie, può trovarla con maggiore immediatezza (1).

Glimpse possiede una data dashboard per l'esplorazione e l'estrazione di dati, dal ricco elenco di sorgenti a cui offre l'accesso. La ricerca nel database avviene attraverso un motore di indicizzazione full-text che fornisce performance di ricerca molto elevate.

La tesi è strutturata come segue. Il primo capitolo presenta una panoramica più dettagliata su Glimpse e sulle tecnologie utilizzate: il database Mongo e l'indicizzatore Elastic Search. Il capitolo illustra inoltre la metodologia di lavoro utilizzata nell'ambito del progetto.

Il capitolo 2 riporta un'analisi dell'applicazione con i casi d'uso per gli scenari che contengono passaggi dei quali ho sviluppato porzioni dell'applicazione.

Dal capitolo 3 inizia la trattazione del lavoro effettivamente svolto durante il periodo di tirocinio. Vengono illustrati gli adattatori sviluppati, prendendo come caso base l'estrazione di dati da OECD ( Organizzazione per la Cooperazione e lo Sviluppo Economico ), e proseguendo il discorso su altri due adattatori, rispettivamente per la ECB ( Banca Centrale Europea ) e l'IMF ( Fondo Monetario Internazionale ). Infine, si trova un ultimo interfacciamento che ha avuto esito negativo riguardante la banca dati delle Nazioni Unite.

Il capitolo 4 illustra nello specifico l'indicizzazione con Elastic Search a confronto con Whoosh, libreria nativa di Python. Vengono quindi illustrati gli algoritmi utilizzati per la costruzione dell'indice, prendendo ad esempio sempre l'algoritmo per OECD, e proseguendo con gli indici delle altre fonti da me sviluppate, ovvero DOE, IMF ed ECB.

A seguire, nel capitolo 5, si elencano gli exporter in diversi formati da me implementati, a partire dal formato base per le immagini, PNG, per poi parlare più ampiamente del formato SDMX, un particolare tipo di JSON, nato da un progetto a cui aderiscono molte delle fonti a cui ci siamo interfacciati e, in particolare, da OECD, che fornisce i suoi dati proprio in questo formato. Infine, l'ultimo formato è quello nativo di Glimpse, ovvero un particolare tipo di JSON, con cui MongoDB mantiene i dati.

Successivamente, nel sesto capitolo, si parla delle modifiche grafiche per migliorare la fruizione e la consultazione dei dati all'utente, dell'inserimento della documentazione che fornisce indicazioni su come utilizzare Glimpse e informazioni per gli sviluppatori che desiderano interfacciarsi.

Il capitolo 7 riporta le conclusioni di questa attività di tirocinio, parlando degli obiettivi prefissi, di quelli ottenuti e dell'arricchimento professionale maturato in questo periodo.

# Indice

Introduzione .....	4
Indice.....	7
1 Il prodotto, le tecnologie e il metodo di lavoro.....	8
1.1 Glimpse .....	8
1.2 MongoDB .....	10
1.3 Elastic Search.....	11
1.4 Scrum, il metodo di lavoro.....	12
2 Analisi dell'applicazione .....	17
2.1 Import dei dati.....	17
2.2 Indicizzazione per ricerche full-text .....	18
2.3 Traduzione per l'export.....	20
3 Gli adapter.....	22
3.1 Il formato dei file .....	22
3.2 OECD.....	24
3.3 ECB e IMF via Quandl .....	28
3.4 UN.....	29
4 Indicizzazione dei dati .....	32
4.1 Elastic a confronto con Whoosh .....	32
4.2 I vari approcci per OECD .....	33
4.3 Indicizzatori per altre fonti.....	40
5 Export delle serie .....	43
5.1 PNG .....	43
5.2 SDMX adattato a Glimpse .....	44
5.3 JMO, il formato di storage.....	46
6 Usabilità .....	48
6.1 La documentazione di Glimpse .....	48
6.2 Miglioramento della resa grafica .....	50
7 Conclusioni .....	53
8 Bibliografia e sitografia .....	54

# Capitolo 1 - Il prodotto, le tecnologie e il metodo di lavoro

In questo capitolo si presenta una panoramica più dettagliata su Glimpse e le tecnologie ad esso applicate, ovvero il database Mongo, utilizzato per la conservazione dei dati, e l'indicizzatore Elastic Search.

## 1.1 Glimpse

Glimpse è un web provider di dati economici e finanziari, o time series, provenienti da diverse fonti in tutto il mondo.

Il portale si suddivide in 4 sezioni, delle quali la prima permette la ricerca delle time series inserendo un identificatore o delle parole significative. Questa seconda immissione comporta una ricerca tramite l'indice, che riporterà quindi più risultati tra cui scegliere.

Una volta trovata la time series desiderata, si possono ottenere alcune informazioni significative, dette *metadati*, per ispezionare la serie prima di effettuarne il download effettivo:

- Nome della serie
- Breve descrizione
- Fonte del dato
- Inizio delle osservazioni
- Fine delle osservazioni
- Frequenza
- Stagionalizzazione
- Data dell'ultimo aggiornamento
- Discontinuità della serie

A questo punto, si può selezionare un *vintage* (se la fonte lo permette) tra quelli disponibili per il dato, ovvero la data in cui la serie è stata calcolata, e passare al Download effettivo, che consiste nel download e caricamento o aggiornamento su MongoDB. Ovviamente con vintage diversi si avranno osservazioni più o meno



coincidenti tra di loro: ad esempio, impostando il vintage all'anno 2013, non si avranno dati disponibili per il 2014, al contrario, selezionandolo all'anno 2014, potrebbero essere emersi nuovi fattori che hanno provocato un ricalcolo per i valori relativi all'anno 2013.

Nella seconda sezione del sito web, l'utente può definire dei dataset, gruppi di più time series organizzate in tabelle, utilizzando le serie scaricate e personalizzandoli con un nome.

Nella successiva sezione, *Transform*, si possono effettuare trasformazioni matematiche sulle serie, e vedere subito il risultato cliccando l'icona di un grafico.

Quando l'utente è soddisfatto dei dati recuperati, organizzati e trasformati, può utilizzare la sezione di *Share* per scaricare le singole time series o i dataset, scegliendo tra le trasformazioni effettuate nella sezione precedente, o lasciando il dato *Raw*, grezzo, ovvero senza trasformate. Si può quindi selezionare il formato preferito per l'export e infine cliccando "Download" si ottengono subito i dati sul proprio PC.

## GLIMPSE

Short time to research delivered



COLLECT

DEFINE

TRANSFORM

SHARE

Timeseries name	fred/all/aaa
Description	Moody's Seasoned Aaa Corporate Bond Yield
Source	FRED
First date	1919-01-01
Last date	2015-08-01
Frequency	Monthly
Seasonal adjustment	Not Seasonally Adjusted
Updated on	2015-08-06 14:21:09-05

Export time series and datasets

fred/all/aaa

Search datasetSearch timeseries

Format: CSV

Vintage: 2015-10-02

Transformation: Raw

Export timeseries

Figura 1 - Una prima immagine che mostra una delle sezioni del portale Glimpse

È prevista un'autenticazione tramite alcuni tra gli account social più conosciuti, quali Gmail, LinkedIn e Amazon, cliccando le rispettive icone nella parte superiore della pagina. Questo per tenere traccia delle serie recuperate ed elaborate e dei dataset creati, in modo da non perdere il lavoro svolto.

Glimpse quindi evita ai suoi utenti l'impiego di ore preziose nel procacciamento delle serie storiche a loro necessarie fornendo una semplice interfaccia che non solo ricerca al loro posto i dati migliori, ma si occupa anche di uniformarli e quindi renderli subito

consultabili all'utente nel formato da lui preferito, dandogli quindi la possibilità di concentrarsi subito sulla loro elaborazione.

## 1.2 MongoDB

L'uniformità dei dati è resa possibile da MongoDB, un database open source per documenti che fornisce alte performance, alta disponibilità e scalamento automatico (2).

In Mongo ogni record è un documento, la cui struttura dati è composta di coppie chiave-valore.

I suoi documenti sono quindi simili a oggetti JSON, e vengono per l'appunto chiamati BSON. I valori dei campi possono includere altri documenti o anche array di essi.

I vantaggi nell'uso di questi documenti sono molteplici:

- i documenti corrispondono a tipi di dati nativi di molti linguaggi di programmazione
- i documenti integrati e gli array evitano la necessità di effettuare join dispendiosi
- lo schema dinamico favorisce il polimorfismo

MongoDB garantisce supporto per modelli di dato integrati riducendo l'attività di I/O sul database. L'indicizzazione supporta query più veloci e può includere chiavi per i documenti integrati e gli array.

Con la sua capacità di replicazione fornisce:

- *Failover* automatico, ovvero reindirizzazione, in caso di problemi con le richieste al database, su repliche dello stesso.
- Ridondanza dei dati, che evita la perdita di informazioni in caso di failure sul database.

La replicazione è chiamata anche *replica sets*, e si basa su gruppi di servers Mongo che mantengono gli stessi set di dati per la ridondanza.

Quindi per effettuare operazione CRUD (create, read, update e delete), Mongo fa uso di questi documenti JSON-like, con campi chiave-valore simili a mappe, dizionari, ecc..., tipici dei linguaggi di programmazione.

I file di Mongo sono chiamati BSON, e sono un tipo binario di file JSON, con informazioni aggiuntive.

Mongo immagazzina i dati in “*collections*”, collezioni. Una collezione è un gruppo di documenti correlati che hanno un set di indici in comune. Sono analoghe alle tabelle dei database relazionali.

Infine, la scalabilità di Mongo permette di immagazzinare enormi quantità di documenti, necessità fondamentale per il software in questione, essendo esso un provider di dati che si espande proporzionalmente al suo utilizzo (3).

## 1.3 Elastic Search

Una delle tecnologie che fanno la forza di Glimpse è Elastic Search (4), un motore di ricerca open-source sviluppato in Java. Esso offre prestazioni elevatissime per ricerche full text e indicizzazioni di grandi moli di dati, fornendo inoltre una ottima scalabilità. Elastic è divenuto un software molto importante nell’ambito Big Data proprio per la capacità di ricercare, analizzare e mostrare dati contenuti nei documenti in formato JSON, con interrogazioni che avvengono quasi in tempo reale. Come molti altri *engine* di questo genere, si basa su Apache Lucene, libreria anch’essa open-source per il recupero di informazioni.

Elastic ha avuto negli ultimi tempi una diffusione enorme, viene ad esempio utilizzato nella parte *front-end* di Wikipedia per le ricerche svolte dagli utenti, quindi nelle stesse modalità di Glimpse, ma anche da Facebook per effettuare analisi interne, e questo può dare un’idea della sua scalabilità, vista la quantità dei dati posseduti da questi due colossi.

Elastic è organizzato in cluster, a cui viene assegnato un nome, ognuno dei quali possiede una collezione di nodi (servers), che garantiscono le performance appena illustrate.

I documenti sono l’unità base che viene gestita da Elastic, ed essi vengono suddivisi in indici, che possono essere in numero elevato per ogni nodo.

Ma il punto chiave della scalabilità di Elastic consiste nella possibilità che il software offre di suddividere gli indici in parti chiamate *shards*, garantendo inoltre ricerche in parallelo.

Per ovviare al problema dei *failover* con conseguente perdita dei dati, Elastic permette la replica delle *shards* in cosiddette *replica shards*. Ovviamente, per evitare perdite, queste repliche vengono conservate su nodi diversi da quello a cui appartiene la *shard*

di origine. Questa tecnica aumenta ancora il *throughput* parallelizzando le ricerche anche sulle repliche (5).

## 1.4 Scrum, il metodo di lavoro

Per lo sviluppo di software all'interno di un team, c'è ovviamente bisogno di definire delle regole, in modo da mantenere un buon livello di organizzazione, assegnando compiti ai vari membri del gruppo. Nel nostro caso, il paradigma applicato durante il periodo del tirocinio è Scrum.

Scrum è un framework di sviluppo con cui un team elabora prodotti o porta avanti progetti in maniera incrementale e iterativa. Scrum ha degli sprint, la cui durata nel nostro caso è stata settimanale.

All'inizio dello sprint il team sceglie degli *items* (definiti dall'acquirente) da un documento chiamato *product backlog*, e li riporta in un diario, uno per sprint, di cui il *product owner* non ha visione. Questo scarica molto la pressione dal team che è comunque tenuto ad autogestirsi e a portare a termine la *task*, questo è il nome che l'*item* assume nel diario, entro la fine dello Sprint.

Il team si confronta quotidianamente sui passi avanti e sui problemi sorti durante dei brevi incontri chiamati *Daily Scrum*.

Scrum enfatizza l'importanza di un prodotto funzionante alla fine di ogni Sprint, nel caso del software, un modulo implementato, funzionante, testato e documentato, potenzialmente inviabile.

Si possono identificare 3 figure per lo *Scrum Team*:

- *Product Owner*: vuole massimizzare il guadagno identificando le *features* prioritarie e modificando continuamente la lista nel *products backlog*.
- *Team*: il team di sviluppo costruisce il prodotto indicato dal *product owner* e deve essere cross-funzionale per rendere fruibile il prodotto realizzato ad ogni sprint, ha molta libertà di concentrarsi su un item piuttosto che un altro ad ogni sprint.
- *Scrum master*: lo *Scrum Master* aiuta in primo luogo il team a lavorare con Scrum, ma anche il *Product Owner*, il tutto per realizzare il prodotto migliore. Inoltre consiglia il team nelle scelte per incrementare la produttività.

Nel nostro caso, il ruolo del *Product Owner* è stato ricoperto da Eleonora Laurenza, che ci ha illustrato inizialmente gli obiettivi che Glimpse cerca di raggiungere, illustrati nell'introduzione, ed ha tenuto aggiornato il *backlog*, collaborando con Luigi Bellomarini, lo *Scrum Master*, che ha coordinato il lavoro del team ed è intervenuto ove necessario con consigli e spunti per l'approccio da seguire durante lo sviluppo. Anche il team, composto da me e da Marco Faretra, mio collega, ha partecipato in alcuni casi alla formulazione di *item* da inserire, come ad esempio nel *refactoring* delle funzionalità del *vintage*, che ha portato ad una riassegnazione di alcune responsabilità all'interno dell'applicazione.

Il *product backlog* è un diario indispensabile all'applicazione di Scrum. Esso evolve durante lo sviluppo del progetto. Serve a mostrare "tutto quello che può fare il team sempre", in ordine di priorità. In questo *backlog* si possono trovare moduli da implementare, o consigli, come risolvere dei difetti o trovare soluzioni per velocizzare processi.

Un buon *product backlog* è DEEP:

- Dettagliato in maniera appropriata: gli *items* con priorità più alta avranno descrizioni più dettagliate, poiché probabilmente verranno lavorati prima di quelli con priorità minore, che avranno descrizioni più generiche, ma solo per il momento.
- Stimato: Ogni *item* viene stimato frequentemente, in base alle nuove informazioni, dal team che stabilisce gli sforzi necessari alla sua realizzazione, dal *product owner* che fornisce informazioni sui rischi e sul valore dell'*item*.
- Emergente: il *backlog* è regolarmente ridefinito, con l'aggiunta, la rimozione e la ridefinizione della priorità degli *items*.
- Prioritizzato: ogni *item* ha una priorità nel *backlog*, e solitamente la priorità maggiore è data all'*item* con maggiore valore d'affari per minor costo.

L'output di uno sprint è un "incremento potenzialmente inviabile del prodotto". Tutte le cose da fare per rendere un *item* inviabile vanno realizzate durante lo Sprint. Inizialmente un team non è capace di fornire un prodotto inviabile ad ogni Sprint, quindi deve lavorare sulla sua automazione e sulla sua cross-funzionalità, in modo da raggiungere questo obiettivo.

Prima di iniziare a lavorare, va stabilita una "Definizione di fatto", una serie di attività che garantiscono l'inviabilità del prodotto. Nel nostro caso, il prodotto è considerato completo rispettando questi punti:

- elaborati di analisi orientata agli oggetti con SSD
- elaborati di progettazione orientata agli oggetti con DCD
- codice sviluppato
- test automatizzato e ripetibile sviluppato
- test eseguito con successo
- documentazione scritta o codice adeguatamente commentato
- codice integrato nel software
- *commit* delle modifiche su *git*

Ogni giorno ci siamo incontrati per il *Daily Scrum*, che sincronizza i membri del team. Ognuno informa gli altri membri su:

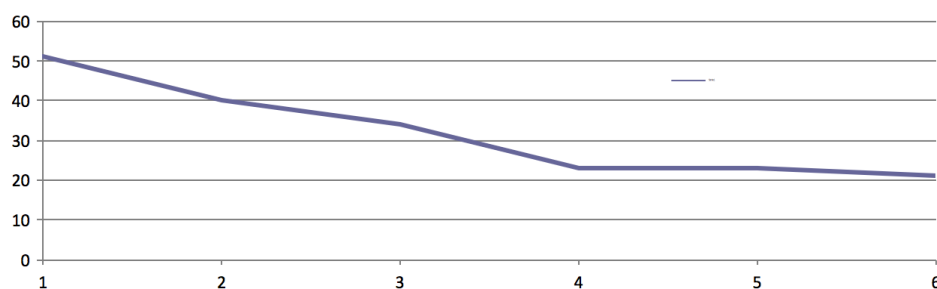
- Cosa è riuscito a fare dall' ultimo incontro
- Cosa farà prima del prossimo incontro
- Quali ostacoli sono sorti

È un modo per informare il team dei propri traguardi o essere aiutati riguardo i problemi sorti. Se necessario viene seguito da riunioni più mirate. In questo modo il team si sente meno pressato, poiché non c'è da rendere conto a un manager, ma è un momento di incontro per migliorare il proprio lavoro.

Il team si gestisce da solo, quindi bisogna sapere come farlo. Ogni giorno, ognuno deve modificare le ore previste per portare a termine le proprie *tasks*. Esso è supportato da un grafico (Sprint Burndown Chart) complessivo delle ore rimanenti alla fine delle *tasks* di tutto il team.

Questo non mostra al team gli sforzi e il tempo speso fino a quel momento, ma il tempo rimanente da lavorare per raggiungere l'obiettivo (6).

Ecco un esempio del grafico, di uno Sprint iniziale, in cui non siamo riusciti a rispettare i tempi previsti durante il Planning. Nel periodo iniziale abbiamo avuto spesso problemi di questo genere per la difficoltà di approccio al software e al metodo di lavoro stesso, magari scegliendo troppe *tasks* rispetto alle proprie possibilità.



*Figura 2 - Uno Sprint Burndown Chart in cui non abbiamo terminato il lavoro*

Alla fine di ogni settimana, il team si riunisce per effettuare la Revisione dello Sprint. In questi meeting si possono porre domande sullo Sprint in questione, ed esso viene analizzato. Vengono però presentate anche le migliorie implementate nel prodotto.

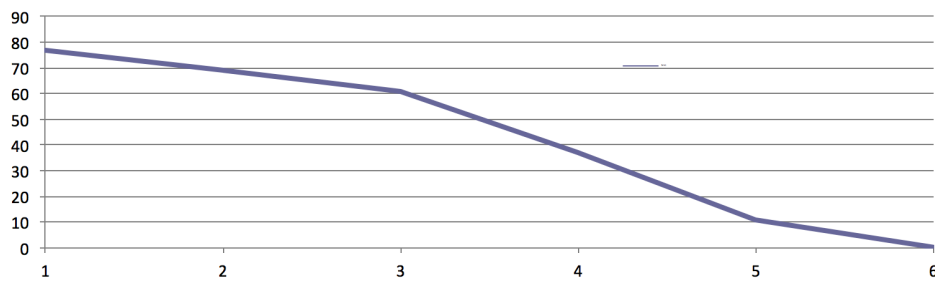
Dopo lo *Sprint review*, il *product owner* si deve occupare di tenere aggiornato il *product backlog*. Solitamente il meeting per il prossimo Sprint avviene il giorno dopo la *review* o comunque nei giorni successivi, in modo da non appesantire il carico sul team. Questo porta ad una maggiore produttività, che potrebbe invece essere rovinata da uno sforzo eccessivo richiesto al team.

Scrum serve a garantire prima di tutto trasparenza. Se al primo Sprint il team non riesce a rispettare le previsioni effettuate, non va preso come un fallimento, ma come un segnale che impone di adattare e magari ridurre il carico, in modo da avere un avvio più lento ma comunque produttivo e in grado di fornire *items* migliori alla fine degli Sprints.

Alcune pratiche non toccate da Scrum possono essere aggiunte a supporto di esso. Scrum non scoraggia l'adozione di strategie quali chiedere consiglio a persone più esperte, ma lascia al team la scelta di queste eventuali soluzioni per velocizzare/migliorare il prodotto degli *Sprints*.

Inizialmente può essere difficile familiarizzare con Scrum, ma verso la fine del progetto si capisce quanto esso può aver facilitato il processo di sviluppo.

Infatti, andando ad analizzare uno degli ultimi Sprint, si può notare come siamo migliorati ognuno nello svolgere le proprie task, anche consigliandoci a vicenda, e riuscendo a portarle tutte a termine.



*Figura 3 - Questo Sprint Burndown Chart riporta uno Sprint terminato con successo*

Questo è stato possibile anche grazie all'aumento della durata degli incontri giornalieri, reso necessario anche dalle funzionalità aggiunte in corso d'opera al software, che hanno richiesto il confronto tra i membri del team per scegliere le strategie migliori da applicare e la risoluzione di eventuali *bug*.



## Capitolo 2 – Analisi dell'applicazione

In questo capitolo viene riportata un'analisi dei problemi di cui mi sono occupato, ed in particolare la raccolta dei dati, la loro indicizzazione ed infine l'elaborazione in altri formati con conseguente export.

### 2.1 Import dei dati

Come abbiamo visto, varie organizzazioni conservano ed offrono agli utenti serie storiche di vario genere, fornendo metodi per usufruirne più o meno semplici, ma anche interfacce per programmatori che vogliano fare da tramite e progettare, come nel nostro caso, portali o software per semplificare la ricerca.

Esistono sorgenti che hanno chiavi univoche, le quali costituiscono il nome della serie. Ad esempio "GDP" in FRED è una singola serie.

Esistono invece sorgenti che hanno chiavi non univoche all'interno della sorgente. Ad esempio "NY.GDP.MKTP.CD" in World Bank indica il GDP di un paese qualsiasi e per queste sorgenti la chiave diventa univoca all'interno di un paese.

Ciò che si propone Glimpse come primo e principale obiettivo, tenendo in conto quanto appena detto, offrire queste serie e scaricarle per gli utenti.

Per portarlo a termine si è resa necessaria a priori un'analisi del dominio e dei requisiti, per definire le responsabilità all'interno del dominio e una struttura per il procedimento nel migliore dei modi, come si richiede usualmente nello sviluppo di software.

Si ha quindi un caso d'uso generale che definisce il processo di ricerca di una serie tramite identificatore:

1. L'utente inserisce una stringa del tipo "paese/nome serie" oppure "nome serie" e sceglie una sorgente S
2.
  - 2.1. Se S ha chiave univoca, allora Glimpse costruisce una richiesta utilizzando solo "nome serie", indipendentemente dal fatto che l'utente ha specificato "paese/nome serie" o "nome serie", eventualmente scartando "paese"
  - 2.2. Se S è di tipo "paese/chiave", allora:

- 2.2.1. Se l'utente ha inserito una stringa di tipo "paese/nome serie", Glimpse usa "paese" e "nome paese" nei campi opportuni della richiesta
- 2.2.2. Se l'utente ha inserito una stringa di tipo "nome serie", allora è interpretata sicuramente come ricerca per descrizione (vedi Indicizzazione)
- 3.
  - 3.1. Se la ricerca per chiave dà una serie come risultato, questa viene visualizzata e preparata per il download
  - 3.2. Se la ricerca per chiave non dà alcun risultato, allora si passa alla ricerca per descrizione.

Per ogni source la costruzione della richiesta per la singola time series tramite le API è variabile, e più o meno complesso in base a come queste interfacce siano state sviluppate.

Si deve quindi ragionare caso per caso sulla documentazione fornita dalle fonti, e trovare soluzioni in base al problema che si tenta di risolvere, sviluppando l'importatore adatto alla situazione.

Il problema in questione è quindi trovare ogni volta una strategia da applicare per le varie fasi di questo processo, inizialmente per ottenere i dati, che può variare nei metodi in cui essi vengono resi disponibili, solitamente tramite chiamate REST, ma alle volte anche in maniera più complicata, come nel caso di quelle SOAP.

Nel caso più diffuso, quello delle REST, che permettono di effettuare query tramite il protocollo HTTP, esse possono avere diversa complessità di costruzione, e anche la completezza della documentazione influisce sulla comprensione del funzionamento delle API.

A questo punto, bisogna analizzare il formato in cui si ottengono le risposte, studiarne la struttura e trovare dove esse riportano i dati significativi che intendiamo recuperare. Solitamente il dato recuperato è in formato JSON, di facile interpretazione, ma in alcuni casi anche esso può assumere strutture particolari e complicate, come nel caso della OECD, con l'SDMX-JSON.

Infine, una volta identificati i dati significativi nella struttura, essi vengono estratti e salvati.

## 2.2 Indicizzazione per ricerche full-text

In caso la ricerca per id non abbia dato risultati, Glimpse passa direttamente a quella per descrizione, senza richiedere altro all'utente.

Lo scenario di questa operazione è illustrato dal seguente caso d'uso:

1. L'utente inserisce una descrizione da ricercare e indica una sorgente
2. Il sistema effettua la richiesta e visualizza l'elenco delle serie risultato
3. L'utente seleziona, navigando nella *scroll bar*, una serie, cliccando sul rispettivo nome
4. Il sistema mostra al posto dell'elenco delle serie la classica tabella con i metadati di quella selezionata, dopo averne effettuato la ricerca per id
5.
  - 5.1.1. L'utente ha letto tutti i metadati della serie, clicca su "Torna indietro"
  - 5.1.2. Il sistema mostra nuovamente l'elenco con *scroll bar* della ricerca fatta precedentemente
  - 5.2.1. L'utente clicca su "Download"
  - 5.2.2. Lo scaricamento avviene e ritorna la schermata con i metadati

Dalle varie fonti, è possibile solitamente estrarre le time series solo conoscendo il loro identificatore. Fanno eccezione alcune API, che permettono anche una ricerca full-text nei database, e quindi dando la possibilità di utilizzare parole chiave. Tuttavia, questo metodo non è ovviamente applicabile a livello generale, ma solo in casi ristretti. Ciò ha portato all'idea di indicizzare i dati all'interno dell'applicazione, senza però scaricarli, creando quindi un grande indice che riportasse identificatore e descrizione per le singole serie storiche. Esso viene interrogato nello svolgimento del caso d'uso appena descritto.

C'è quindi la necessità di sviluppare un indice interrogabile dal software che possa trovare e ordinare in base a un punteggio la corrispondenza alle numerose serie partendo dall'input dell'utente. Per fare questo abbiamo utilizzato, come detto precedentemente, Elastic Search.

La costruzione di questo indice, per ogni fonte, ha richiesto, per cominciare, ovviamente, tutte le serie da essa fornite, o almeno i valori necessari all'inserimento nell'indice, ovvero identificatore e descrizione.

Si rende quindi necessaria l'implementazione di un indicizzatore per ogni fonte, che generi da ogni particolare tipo di input un output compatibile con i parametri imposti da Elastic e ne effettui poi il caricamento su quest'ultimo.

La raccolta dell'input è quindi il passaggio che richiede approcci diversi, in base ai metodi offerti dalle fonti per risalirvi.

Come vedremo, in alcuni casi, si sono provati diversi algoritmi per avere risultati migliori in termini di tempo relativamente alla costruzione di questi indici.

## 2.3 Traduzione per l'export

Il passaggio conclusivo che effettua Glimpse consiste nel garantire la possibilità all'utente di scaricare i dati raccolti, garantendo diversi formati di output, seguendo dei passaggi illustrati da questo caso d'uso:

1. L'utente accede al pannello di esportazione
2. Il sistema propone i dataset definiti e le time series scaricate
3.
  - 3.1.1. L'utente inserisce il nome di un dataset
  - 3.1.2. Il sistema mostra tutte le serie del dataset e, per ciascuna, le eventuali trasformazioni eseguite e i vintage scaricati
  - 3.1.3. Per ogni serie l'utente può scegliere trasformazioni e vintage o lasciare il default
  - 3.2.1. L'utente inserisce l'identificatore di una time series
  - 3.2.2. Il sistema mostra i metadati della time series, le eventuali trasformazioni eseguite e i vintage scaricati
  - 3.2.3. L'utente seleziona trasformazione e vintage desiderati
4. Il sistema propone una serie di formati di esportazione
5. L'utente seleziona un formato e conferma
6. Il sistema conferma l'esportazione

Il successo di questo scenario è determinato soprattutto dalla corretta traduzione dei file salvati nel database interno di Glimpse, si deve quindi conoscere bene la struttura iniziale e quella di destinazione, in modo da implementare un algoritmo con cui l'esportatore riesca a restituire un output corretto e senza perdita di dati significativi della serie.

Essendo JSON il formato di partenza, viene saltato il passaggio iniziale di studio della struttura di partenza, data la sua estrema semplicità, che ha garantito la sua enorme diffusione.

Nel caso di formati di destinazione meno conosciuti o comunque più complessi, la traduzione risulta più complessa, per l'assenza di supporti quali librerie o moduli, come ad esempio nel caso dell'export in formato SDMX-JSON. In questi casi, si devono studiare le documentazioni e le strutture. È molto utile anche utilizzare esempi di file che possano chiarire meglio l'organizzazione del formato e quindi possano dare una base per definire una struttura adattiva alle serie storiche e, quando possibile, ai dataset.

## Capitolo 3 – Gli adapter

In questo capitolo si inizia a parlare del lavoro svolto, illustrando il ruolo degli adattatori in generale e, nello specifico, quelli da me sviluppati, partendo da quello per la OECD, e in seguito parlando di quelli ottenuti tramite Quandl per la ECB e l'IMF. Viene infine descritto l'approccio all'interfacciamento con UN, l'Organizzazione delle Nazioni Unite, non andato a buon fine per motivi esterni.

L'applicazione prevede una superclasse principale "Importer", ereditata dai singoli adattatori, che offre alcuni metodi comuni, come quello per le chiamate REST e per la pulizia del titolo, e ne definisci altri lasciando l'implementazione alle classi figlie.

L'Importer del caso, legato alla fonte specificata, viene creato da un "ImporterFactory", a sua volta inizializzato nella "view".

Quest'ultima si incarica poi di richiedere all'adattatore appena istanziato dati o metadati, in base all'operazione effettuata dall'utente. Per ognuno di questi Importer si ha poi una diversa implementazione della raccolta delle time series.

Ogni fonte che possiede database di serie storiche, le rende pubbliche e consultabili, fornendo inoltre delle API per gli sviluppatori, dando loro la possibilità di sviluppare software in grado di interrogare queste banche dati e successivamente estrarli, solitamente tramite chiamate REST. Esse vanno quindi comprese caso per caso e utilizzate al meglio per estrarre i dati, che vengono poi salvati sul database Mongo di Glimpse.

### 3.1 Il formato dei file

Mongo assegna un id univoco ai documenti, insieme agli altri campi definiti. Nel nostro caso, i dati assumono due formati, quello delle *time series* e quello dei *datasets*. Con il primo vengono salvate, appunto, le serie singole, e possiede un array di dizionari "data", ognuno con 5 campi:

- *date*, che riporta la data effettiva dell'osservazione, è uno dei due elementi base della serie storica
- *tr*, la trasformazione a cui è legata la time series in questione, che può essere "raw" se si tratta di quella base, "grezza", oppure assume il nome della particolare operazione effettuata sulla serie, ad esempio "log"

- *realtime\_start*, che riporta l'estremo inferiore dell'insieme rappresentante il periodo di tempo in cui l'osservazione è ritenuta valida
- *realtime\_end*, il quale riporta l'ultima data di validità per l'osservazione. Se quindi è riportata una data precedente alla data odierna, si implica la possibile esistenza di una versione della variabile più aggiornata che potrebbe avere valore diverso
- *value*, che contiene il valore effettivo dell'osservazione, è l'altro valore base della serie storica. L'unione di questi permette l'estrapolazione di un grafico che definisce l'andamento dell'argomento rappresentato

Questo array è affiancato da un dizionario, “*metadata*”, con 13 campi, in questo ordine:

- *id*, che riporta l'identificativo da noi utilizzato in Mongo per i documenti, nel formato *fonte/nazione (all se non specificata)/id\_time series*
- *last\_updated*, data di ultimo aggiornamento della time series
- *observation\_start*, la data minore delle osservazioni, in poche parole riporta quando si è iniziato a tenere traccia dell'informazione in questione
- *title*, che riporta la descrizione della time series, nella quale in alcuni casi si possono identificare alcuni dei valori della serie, quali la frequenza e l'unità di misura, in base al livello di dettaglio offerto dalla fonte che la espone
- *seasonal\_adjustment*, per la stagionalizzazione
- *observation\_end*, la data finale delle osservazioni
- *realtime\_start*, che riporta il valore più piccolo tra le osservazioni in “*data*”
- *realtime\_end*, il quale riporta il valore maggiore in “*data*”
- *discontinued*, che indica se la serie ha discontinuità
- *source*, la fonte che ha fornito il dato in questione
- *frequency*, la cadenza delle osservazioni, che ha come granularità più piccola la giornaliera
- *units*, unità di misura utilizzata per i valori, se esistente, reperibile o non scontata
- *source\_name*, aggiunto da me per l'implementazione del formato SDMX (vedi capitolo 5), che riporta il nome completo della fonte

L'id nei metadati coincide quindi con l'identificativo front end che utilizza l'utente per reperire la time series, o con parte di esso. La fonte è infatti omettibile durante la ricerca perché impostata dal menù a tendina.

Il secondo formato riporta i datasets creati dall'utente, e consiste in un array il cui nome coincide con quello assegnato al dataset al momento della sua creazione, mentre i valori coincidono con gli id delle time series.

```
{
  "_id" : ObjectId("55b29d6e8dbd886775e83ae3"),
  "pippo" : [
    "FRED/ALL/AAA",
    "FRED/ALL/GDP",
    "ECB/ALL/RTD_M_S0_N_P_OILBR_E"
  ]
}
```

Per la consultazione di Mongo ho utilizzato il software *Robomongo*, che permette la navigazione del database e la ricerca tramite semplici query preimpostate nelle quali basta inserire il testo ricercato.

Esso mi è stato utile nel confronto dei dati successivamente allo sviluppo degli importatori (capitolo 3) e anche come supporto aggiuntivo per controllare la correttezza dei file da esportare (capitolo 5).

## 3.2 OECD

La OECD, Organizzazione per la cooperazione e lo sviluppo economico, offre un ricco database di oltre 7 milioni di differenti time series, organizzate in più di 1000 datasets, che nel loro caso rappresentano una suddivisione per argomento. Ognuno di essi ha infatti una breve descrizione che definisce l'ambito delle serie in esso riportate.

OECD offre 2 tipi di query, una per ottenere la struttura dei datasets, e un'altra per estrarre da essi le time series, conoscendone l'id.

Quest'ultime possiedono un determinato numero di dimensioni, relativamente al dataset che le contiene, che possono rappresentare la località, la frequenza, la materia del dato, la sua unità di misura, il donatore o il ricevente di un determinato bene o servizio, ecc. Inizialmente la loro varietà mi ha confuso, sia perché è stata la mia prima *task* di interfacciamento, sia perché per alcuni dataset non era possibile reperire la



localizzazione del dato tra le dimensioni, impedendo la suddivisione delle time series nel formato di Glimpse (*fonte/paese/id*).

## La documentazione

I dati forniti da OECD (7) hanno, per le due differenti query, rispettivamente output in formato XML e SDMX, un tipo di JSON. Comunque, per questa prima parte di lavoro relativa a OECD, solo la seconda è stata necessaria, poiché ho stabilito insieme al team come requisito per l'utente la conoscenza del dataset e del numero di dimensioni significative ad esso legate. L'utente è quindi familiare alle serie di OECD, e inserisce l'identificativo del dataset e le dimensioni che identificano la time series da lui richiesta.

Ad esempio, volendo estrarre da OECD, dal dataset QNA, che riporta i “Conti Nazionali Quadrimestrali”, i dati sul Prodotto Interno Lordo (B1\_GE) dell'Austria (AUS) nella valuta nazionale (CARSA), calcolati annualmente (A), l'utente inserirà nella barra di ricerca il seguente valore:

- ALL/QNA.AUS.B1\_GE.CARSA.A

OECD fornisce anche altri parametri opzionali da inserire nelle query, quali i lassi temporali del quale estrarre il dato, ma questa feature non è offerta da Glimpse, quindi non ne ho fatto uso. Invece, la possibilità di specificare l'estrazione dei soli metadati dalle query, è stata utile per velocizzare l'estrazione della time series, ma, come vedremo, lo è stata soprattutto nella creazione dell'indice.

## Estrazione dei metadati

Passando all'implementazione, l'utente passa alla *view* i dati relativi alla time series richiesta di OECD, a questo punto l'*ImporterFactory*, rilevando la *source*, istanzia il relativo adattatore, *OECDImporter*. A questo punto, viene evocato il suo metodo per la raccolta dei metadati, che per prima cosa identifica e recupera il dataset e le relative *dimensions*, salvandoli in due variabili, “*dataset*” e “*values*”. Con essi successivamente costruisce la query, e viene effettuata la chiamata REST, ereditata dalla classe *Importer*.

A questo punto la risposta alla chiamata, viene tradotta in JSON, essendo essa una stringa formattata in SDMX.

Questa struttura permette di risalire facilmente alle date delle osservazioni, tutte riportare separatamente dai valori, ma formattate in diverse maniere, in base alla granularità, che va dal giornaliero (aaaa/mm/gg) all'annuale (aaaa).

Questo ha reso necessaria l'implementazione di un controllo sulla data, dopo averla estratta dal file, che riunificasse il formato delle date a quelle utilizzate da Glimpse, in modo da non avere incongruenze al momento dell'utilizzo di esse.

Per impostare la descrizione della time series ho unificato quelle del dataset e delle dimensioni, in modo da avere la certezza di aver selezionato quella corretta.

A questo punto, dopo aver salvato i metadati in una variabile, si rende necessario un controllo sull'esistenza dell'unità di misura e della frequenza, che come abbiamo detto, non sono sempre presenti nei dataset. Il controllo viene effettuato utilizzando parole chiavi per entrambe, cercandole negli id di tutte le dimensioni, finché non viene trovato l'id corrispondente, altrimenti nei metadati viene riportata la non presenza di unità di misura o frequenza, che per particolari datasets di OECD può essere ammissibile.

A questo punto viene effettuato anche un controllo sulla stagionalizzazione della serie, ricercando parole significative nel corpo della serie.

Infine, i valori della time series che verrà restituita all'utente dalla vista, vengono riempiti utilizzando i metadati appena estratti dalla risposta.

```
ts.metadata["observation_start"] = obs_start
ts.metadata["observation_end"] = obs_end
ts.metadata["realtime_start"] = obs_start
ts.metadata["realtime_end"] = datetime.datetime.now().strftime("%Y-%m-%d")
ts.metadata["title"] = self.clean_title(self.remove_copyright(title))
ts.metadata["frequency"] = frequency
ts.metadata["units"] = units
ts.metadata["last_updated"] = datetime.datetime.now().strftime("%Y-%m-%d")
ts.metadata["seasonal_adjustment"] = seasonal_adjustment
ts.metadata["source"] = "OECD"
ts.metadata["source_name"] = "Organisation for Economic Co-operation and Development"
ts.metadata["discontinued"] = "No"
ts.clean_metadata()
```

## Organizzazione dei dati

Per quanto riguarda l'estrazione effettiva della time series, quindi delle misurazioni, si procede analogamente a quella dei metadati, costruendo l'URI con dataset e dimensioni, ed effettuando la chiamata REST.

Viene salvata in una variabile la lista delle date delle osservazioni, che nei file SDMX si trova in una parte diversa rispetto alle osservazioni.

A questo punto, vengono estratti dalla risposta i dati, conservati in un array indicizzato da numeri, in quantità uguale alla somma delle dimensioni più uno ulteriore per le osservazioni, frapposti da “.”.

Per chiarire, se il dataset ha ad esempio 3 dimensioni, le chiavi assumono la forma “x:y:z”, dove ognuno dei numeri cambia a seconda della dimensione a cui corrisponde in base al suo valore. Ovviamente, è probabile che non tutte le combinazioni esisteranno all’interno del dataset.

Ad ognuno dei numeri quindi corrisponde una data dimensione, identificabile in un’altra area del file SDMX. Nel nostro caso, ognuno ha solo valore 0, poiché la time series estratta è una sola, e l’unico numero a cambiare è l’ultimo, quello delle osservazioni, che appunto saranno molteplici. Per ordinare questo particolare tipo di indice, ho sviluppato una comprensione di liste che seleziona ognuno dei numeri e ordina la lista, in modo da ritornare le osservazioni in serie, dalla prima alla più recente. Quindi, viene identificata la dimensione legata alle osservazioni, per estrarne i valori ed inserirli in una lista che riporta i dati nelle modalità viste all’inizio del capitolo.

A questo punto, la lista viene assegnata alla time series che verrà restituita e già possiede i metadati.

## Revisione dell’approccio

Ricapitolando, per l’estrazione dei metadati:

- Data la richiesta in input dell’utente, la *view* incarica l’*ImporterFactory* della costruzione di un’istanza di *OECDImporter*
- All’*importer* appena istanziato viene passato l’input, che utilizza per costruire la *query* definita da OECD per i dataset specificando ogni dimensione ad esso relativa
- Viene effettuata la chiamata *REST* con il metodo ereditato dalla superclasse *Importer*
- La risposta viene *parsata* in formato JSON
- Viene eseguito il metodo “*time\_reformat*” sulle date estratte dal file per uniformarle
- Vengono ricercate unità di misura e frequenza tra le dimensioni
- Si costruisce il titolo della serie utilizzando la descrizione del dataset e delle singole dimensioni

- Vengono ricercate informazioni sulla stagionalizzazione della serie ispezionando tutto il file
- Infine, si riempiono i metadati della serie da restituire con i dati estratti

Per quanto riguarda l'estrazione dei dati:

- Si effettua la richiesta, traducendola poi in JSON, come visto sopra
- Vengono salvate il dizionario che riporta le osservazioni e la lista delle date a esse relative
- Le osservazioni vengono ordinate dalla meno recente in poi
- Viene identificata la dimensione relativa alle osservazioni e la si utilizza, insieme alle osservazioni, per costruire il dizionario dei dati

### 3.3 ECB e IMF

Lo sviluppo degli adattatori per ECB e IMF è risultato di gran lunga più facile e veloce, dato l'appoggio a Quandl per l'estrazione.

Il procedimento segue gli stessi passaggi di OECD, richiedendo una query REST, con l'aggiunta di una variabile “*API key*”, che Quandl offre agli utenti iscritti e necessaria a non avere limitazioni sulle richieste effettuabili ogni giorno. La cosa che varia da OECD è l'estrazione dei dati significativi dalla risposta, in questo caso molto più immediata data la semplicità della sua struttura.

Quandl è anch'esso un provider di serie storiche, e questo lo rende nostro *competitor*, ma utilizza un approccio differente, creando a priori i suoi databases di dati relativi alle varie fonti, e offrendoli all'utenza (8).

Mi sono quindi interfacciato direttamente ad essi, tramite le API di Quandl, che offrono una dettagliata documentazione e utilizzi veramente semplici (9).

Per ottenere i metadati mi è quindi bastato costruire l'URI indicato nella documentazione ed effettuare la chiamata REST alle API di Quandl, specificando la richiesta dei soli metadati. Ho poi inserito dei controlli sui singoli metadati per evitare che alcuni dei dati forniti, detti indici, potessero creare problemi. Essi sono infatti sprovvisti di metadati e riportano un solo valore specifico nelle osservazioni. A questo punto i dati vengono attribuiti alla time series da restituire all'utente.

Per quanto riguarda l'estrazione dei dati, la query varia di poco, specificando in questo caso la richiesta dei dati, e indicando l'ordine ascendente per il file JSON di risposta in modo da avere i dati in serie dal primo all'ultimo temporalmente parlando, mentre per il caso precedente si è reso necessario l'ordinamento una volta scaricata la serie. I dati sono poi facilmente attribuibili alla time series da salvare nel database per dare la possibilità all'utente di effettuarne l'export.

Gli algoritmi, sia per l'estrazione dei metadati che dei dati, sono identici per le due fonti, poiché cambia solo il database a cui viene effettuata la richiesta, in particolare ECB per la Banca Centrale Europea e ODA per l'IMF. Questo apre alla possibilità di un refactor dei nostri adattatori che forniscono dati tramite Quandl, inserendo una *Factory* e una superclasse *QuandlImporter* da cui i singoli *Importer* varierebbero solo per il database ad essi assegnato per l'estrazione.

### 3.4 UN

L'UN, l'organizzazione delle Nazioni Unite, fornisce i suoi dati tramite il portale *UNData*, nel quale riporta anche delle API per l'interfacciamento (10).

L'interfacciamento richiede requisiti diversi e quindi l'approccio è differente, infatti, in questo caso, viene richiesta come prima cosa la costruzione di un file *SDMX-ML*, una derivazione del formato *XML* definita dal progetto *SDMX*.

Esso serve per definire i parametri della *query* in base alla serie che si intende recuperare. Il file può essere costruito tramite l'"*SDMX browser*" fornito dalla *UNData* stessa (11). In questo sito si seleziona da un menù a tendina sulla sinistra l'argomento desiderato e quindi il dataset. A questo punto sulla finestra a destra appaiono delle etichette che riportano le dimensioni presenti e i loro valori, che possono essere selezionati a piacere. Infine, si può procedere al download della query, ottenendo il file *XML*.

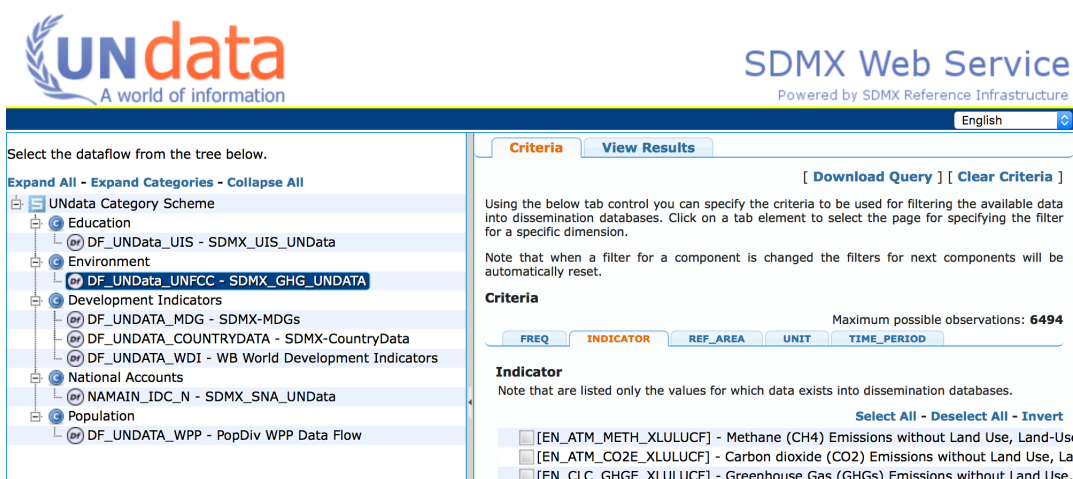


Figura 4 - Interfaccia dell'SDMX browser offerto da UNData

Una volta ottenuto, esso va inviato tramite una richiesta *HTTP* al *WebService* di *UNData*:

- <http://data.un.org/ws/NSISStdV20Service.aspx>

Esso utilizza *SOAP*, un protocollo Web Service basato su *XML*, che sfrutta il protocollo *HTTP* per il trasporto, ma offre metodi per chiamate remote ed esterne al Web. Quindi il file *XML* scaricato andrà incapsulato in un altro *XML* basato su *SOAP*, e infine inviato al *WebService*, che infine restituirà un file dello stesso tipo basato sul *DSD* del dataset selezionato.

Poiché la query in questo caso non è assemblabile come negli altri casi partendo da dati di input e procedendo ad una loro elaborazione, ma solo tramite un'effettiva interazione con l'*SDMX browser* effettuando dei click sulla pagina, ho fatto ricorso ad un software di supporto al *testing* per applicazioni web, *Selenium*. Esso permette a livello di codice l'interazione con i browser, sia in apertura che in utilizzo, simulando click sulla pagina (12).

*Selenium* permette di istanziare i *driver* dei browser, e ad essi viene delegata la ricerca di elementi e il "click" sulla pagina. Ho quindi istanziato un oggetto *driver* per *Chrome* a cui ho delegato l'apertura di una finestra del browser sulla pagina dell'*SDMX browser*. A questo punto, utilizzando una lista inserita manualmente e riportante i nomi dei dataset (visibili nella colonna di sinistra della figura precedente), il driver si occupa di aprirli tutti tramite il metodo "click" offerto da *Selenium*.

Quindi viene effettuato un controllo sul dataset inserito dall'utente, per controllare se esso è effettivamente tra quelli offerti. In caso positivo, in base al dataset selezionato appaiono le etichette delle dimensioni con i rispettivi valori sottostanti selezionabili. A questo punto vengono lette anche le dimensioni inserite, e per ognuna viene cercato e selezionato, se esiste, il valore coincidente.

Finita la definizione della serie, viene cliccato il pulsante "*Download Query*" e il file XML viene scaricato.

Una volta ottenuto quest'ultimo, si utilizzano le strutture *SOAP* offerte dalla documentazione per inviare la richiesta completa al *Webservice*.

Tuttavia la documentazione e il supporto fornitoci da UN sono stati insufficienti per progettare e realizzare una soluzione funzionante. Il task è stato quindi sospeso e rimandato a sviluppi futuri poiché, tra l'altro, siamo stati informati dell'imminente rilascio di un'interfaccia REST da parte di UN, sicuramente di più semplice utilizzo, essendo a questo punto le procedure simili a quelle richieste dalle altre API, con la costruzione di query HTTP.

## Capitolo 4 - Indicizzazione dei dati

In questo capitolo viene illustrata nello specifico l'indicizzazione con *Elastic Search*, dopo un paragone con *Whoosh*, il *tool* adottato inizialmente, nativo di Python, illustrando le motivazioni che ci hanno portato a preferire il primo. Viene quindi illustrato come è strutturato l'algoritmo di costruzione dell'indice per OECD. Successivamente si parla degli indici delle altre fonti da me sviluppati, ovvero DOE, IMF ed ECB.

### 4.1 Elastic a confronto con Whoosh

Inizialmente per la creazione degli indici abbiamo tentato un approccio con *Whoosh*, una libreria di Python. Esso permette ricerche full-text ed è sviluppato interamente in questo linguaggio. Questo lo rende molto affidabile, poiché non necessita l'utilizzo di package o librerie esterne. Esso possiede anche algoritmi di *scoring* per ordinare i risultati in base alla coerenza con la chiave di ricerca e di analisi per il testo.

*Whoosh* necessita della definizione di uno *schema* per la struttura dell'indice, che va effettuata solo al momento della creazione di quest'ultimo.

Questa libreria fornisce dei campi particolari, come l'”*id*”, la “*parola chiave*” o la “*data*”, che corrispondono a vari tipi, quali stringhe e interi, e definiscono cosa si indicizza e cosa è ricercabile.

Una volta creato l'indice, si può iniziare ad aggiungere documenti, con un semplice metodo “*add\_document*” che ha come argomento una serie di coppie chiave valore, dove il secondo è richiesto in *unicode* per i valori indicizzati, mentre per gli altri può essere in qualsiasi formato.

Inoltre, non è necessario passare tutti i valori identificati dallo schema, essi possono essere semplicemente omessi nel documento.

*Whoosh* è quindi molto utile in questi casi:

- Ovunque sia preferibile una soluzione pura in Python per evitare la compilazione di librerie native
- Come piattaforma di ricerca per sviluppatori che preferiscano lavorare esclusivamente con Python piuttosto che anche con Java



- Quando un'interfaccia interamente in Python è più importante della velocità di ricerca (13)

Quest'ultimo motivo, insieme all'inefficiente scalabilità della dimensione dell'indice e della lentezza di creazione di quest'ultimo, ci ha portato a preferire l'utilizzo di Elastic Search, i cui punti forti sono stati illustrati nel capitolo 2, molto più adatto alle nostre necessità, data la quantità dei dati da indicizzare.

Basti pensare che solo con OECD l'indice contiene più di 6 milioni di time series.

## 4.2 I vari approcci per OECD

Come appena detto, il problema fondamentale per questo lavoro sono state le dimensioni del database di OECD, che mi hanno portato in primo luogo a preferire l'utilizzo di *Elastic* piuttosto che di *Whoosh*.

Per iniziare quindi ho costruito una lista di URL di 1005 elementi, uno per dataset, in modo da poter scaricare l'intero database di OECD.

Per la costruzione di questa lista ho utilizzato l'altro tipo di query offerta da OECD, che restituisce la struttura dei singoli dataset, ma sostituendo al nome del dataset la stringa "all", in modo da ricevere la struttura di tutti i dataset:

- <http://stats.oecd.org/restsdmx/sdmx.ashx/GetDataStructure/all>

Questa chiamata restituisce un file XML che ho *parsato* utilizzando la libreria *xml.etree.ElementTree* di Python, la quale semplifica l'interrogazione di questo tipo di file offrendo una semplice interfaccia. Da esso estraggo quindi i nomi dei dataset, inserendoli oltre che nella lista anche in un file che può essere riutilizzato successivamente tramite un metodo apposito evitando la query, e quindi risparmiando il tempo impiegato per essa.

Costruita la lista, c'è la necessità di effettuare una richiesta per ognuno dei dataset che restituisca l'insieme di tutte le time series. OECD permette il download di tutte le serie di un dataset sostituendo nella query alle singole dimensioni la stringa "ALL", come mostrato:

- [http://stats.oecd.org/SDMX-JSON/data/<dataset\\_name>/ALL](http://stats.oecd.org/SDMX-JSON/data/<dataset_name>/ALL)

Avendo necessità dei soli metadati per la costruzione dell'indice, ho sfruttato uno dei parametri opzionali supportati dalle API, “*detail*” che offre varie opzioni legate al tipo di dati estratti dalle query:

<b>detail</b>	<p>This attribute specifies the desired amount of information to be returned. Possible values:</p> <ul style="list-style-type: none"><li>• Full: all data and documentation, including annotations (default)</li><li>• DataOnly: attributes – and therefore groups – will be excluded</li><li>• SeriesKeysOnly: only the series elements and the dimensions that make up the series keys</li><li>• NoData: returns the groups and series, including attributes and annotations, without observations</li></ul>
---------------	--

Figura 5 - Porzione della documentazione fornita da OECD

Utilizzando l'opzione “*NoData*”, ho ridotto la dimensione dei file scaricati, evitando di occupare spazio inutilmente e velocizzando il download (7).

Infatti, il download di alcuni tra i dataset più grandi ha richiesto più di 150 MB di spazio su disco, ridotto di più del 10% con l'utilizzo di questo parametro aggiuntivo, guadagnando anche in termini di tempo.

A questo punto, identificata la migliore struttura per le query, sono arrivato al momento del loro utilizzo.

Ho quindi provato diversi algoritmi per questo obiettivo, tentando abbassare i tempi di esecuzione.

## Approccio concorrente

In questo approccio ho considerato la programmazione concorrente, che permette l'esecuzione di più processi contemporaneamente, con la particolarità di condividere strutture dati, a differenza della normale programmazione con istruzioni eseguite sequenzialmente l'una dopo l'altra, che ammette comunque l'esecuzione di processi contemporanea ma con la condizione che essi non interferiscano.

A questo scopo, ho utilizzato dei moduli di Python adatti allo scopo: *Queue* e *Thread*. Con il primo si ha la possibilità di istanziare una coda, che nel nostro caso è stata logicamente popolata dalla lista delle query ai dataset. La variabile “*concurrent*” modificabile a piacere, che nei test ho variato per trovare valori ottimali, determina il numero di *Thread* istanziati contemporaneamente, ognuno dei quali si occupa

dell'esecuzione del metodo “*doWork*”, che estrae dalla coda un URL, lo utilizza per effettuare la richiesta e successivamente elabora ricevuti nella risposta.

```
global q
print("Starting indexing OECD")
q = Queue(concurrent * 2)
for i in range(concurrent):
    t = Thread(target=doWork)
    t.daemon = True
    t.start()
```

I tempi di download con queste modalità sono risultati veramente bassi, ma hanno portato ad una perdita non sottovalutabile di dataset che non venivano scaricati correttamente e, alzando la quantità di query concorrenti, a volte anche ad un blocco nell'esecuzione dell'algoritmo.

## Approccio retry

Per provare ad ovviare alla errata *response* da parte di alcuni dataset, nel metodo “*get\_body*”, il quale si occupa di effettuare la query e restituirne il risultato, ho inserito un controllo che, nel caso di risposte andate in errore (solitamente il *body* restituito contiene esclusivamente una stringa “None” o “Internal server error”), mette in pausa il *thread* per un tempo predefinito grazie al modulo “*time*” nativo di Python. Dopo l'attesa, la query viene nuovamente eseguita, fino ad “n” volte, stabilite da una variabile incrementata in un ciclo “*while*”. Questa politica ha effettivamente dato risultati migliori, aumentando la percentuale di dataset recuperati: ho notato un miglioramento graduale per ogni *retry* nell'ottenere il dato integro.

Ad esempio, dal *logger* con cui ho monitorato questi *retry*, ho constatato che su circa 200 query che falliscono al primo tentativo, solo 150 arrivano al secondo *retry*, per scendere successivamente a meno di 100, fino a 20-25 circa nell'ultimo, che in questo caso si tratta del quinto.

Tuttavia, questo algoritmo, oltre a garantire una quasi totale ma comunque non completa lista delle serie di OECD, è soggetto a blocchi e a reset da parte dei server della fonte, non adatti a elevati numeri di richieste allo stesso tempo.

Questo ha scoraggiato l'utilizzo della concorrenza e sono passato ad un approccio più lento ma sicuramente più affidabile, eseguendo una query alla volta.

## Approccio download-all-first

Questa volta ho provato a scaricare i dataset in serie, posticipando l'elaborazione dei dati. Ho quindi eliminato la coda e iterato la lista degli *URLs* tramite un semplice “*for*”, dopo averla costruita con le stesse modalità dei precedenti algoritmi.

Con questo approccio, a volte l'algoritmo si blocca sul download, ho quindi aggiunto un timeout per le singole connessioni al metodo “*get\_body*”, effettuate tramite la libreria di Python *httplib*.

Questo approccio è sembrato inizialmente buono, completando il download in circa mezz'ora e con pochi dataset andati in errore, ma incappando anche in questo caso in un reset delle connessioni da parte del server per la gran quantità di richieste in poco tempo.

Inoltre, in questo modo si va anche a creare una grande quantità di dati grezzi nell'ordine dei *GigaByte* che vanno ad occupare spazio sul disco, da elaborare per l'indicizzazione in un secondo momento.

Ho dovuto quindi rielaborare nuovamente l'algoritmo per motivi di inefficienza.

## Approccio download-and-process

Infine, ho deciso di effettuare il download di ogni dataset e processarlo subito, in modo da distanziare in termini di tempo le richieste al server OECD, evitando così il reset delle connessioni.

Quindi, una volta costruita la lista degli *URLs*, essa viene iterata come visto precedentemente ed il numero del dataset viene tenuto da una variabile contatore.

Ad ogni entrata nel ciclo viene chiamato il metodo “*get\_body*” con l'URL corrente come parametro, che si occupa di effettuare la richiesta, restituendo la *response* in forma di stringa o “None” in caso di errori o *timeout* della connessione, anche in questo caso impostata a 30 secondi.

```

def getBody(first_part_url):
    """This method returns a body with all dataset's timeseries,
    None if it times out (timeout default is 30 seconds)"""
    try:
        print("Getting body from: " + first_part_url)
        complete_url = urlparse(first_part_url)
        conn = httplib.HTTPConnection(complete_url.netloc, timeout=30)
        conn.request("GET", complete_url.path+"/all?detail=NoData")
        res = conn.getresponse()
        return res.read()
    except Exception, e:
        with open('glimpse_ts/indexer/oecd_failed', 'a') as fp:
            fp.write(first_part_url + "\n")
        if "socket.timeout" in str(type(e)):
            with open('glimpse_ts/indexer/oecd_timedout', 'a') as fp:
                fp.write(first_part_url + "\n")
            print("timeout with: " + first_part_url)
        else:
            print e
            print("ERROR WITH: " + first_part_url)

```

Come si può vedere, gli errori vengono gestiti tramite eccezione, e non appena viene rilevata, l'URL del dataset viene riportato nel file di testo *“oecd\_failed”*. Quindi, se il problema è derivato dal *timeout*, il dataset viene riportato anche in *“oecd\_timedout”*, che serve successivamente per scrivere il *logger*. Se l'errore è un altro, esso viene stampato a schermo.

A questo punto, il *body* viene analizzato e, se corretto, viene utilizzato per iniziare la costruzione del file *“OECD.elastic”*. Questo file consiste in un file di testo formattato secondo le *“API Bulk”* (14) di Elastic. Esse permettono il caricamento di grandi quantità di documenti, migliorando enormemente in termini di tempo rispetto all'inserimento singolo in serie.

Il caricamento tramite Bulk richiede un file di testo formattato in JSON, consistente in una coppia di stringhe per ogni documento, la prima indicante l'azione da eseguire, nel nostro caso *“create”*, affiancata da valori quali il nome dell'indice, il tipo (sottoindice) e l'id per Elastic, e la seconda relativa ai dati da salvare nel documento, nel nostro caso quindi *“id”*, *“name”*, *“description”*, *“source”* e *“country”*:

```

{"create":
  { "_index": "timeseries", "_type": "external", "_id": "oecd.all.sna_table1.irl.b1g.c"
  }
}
{"id": "oecd/all/sna_table1.irl.b1g.c",
"name": "1. Gross domestic product (GDP)",
"description": "1. Gross domestic product (GDP) and Ireland and Gross value added at basic prices, total activity and Current prices",
"source": "OECD",
"country": "ALL" }

```

La creazione di questo file è delegata al metodo *“build\_elastic\_file”*, a cui viene passata la *response* di *“get\_body”*. Essa è in formato SDMX, un particolare tipo di

JSON nato da un progetto ancora in corso a cui contribuiscono varie organizzazioni nazionali e internazionali, tra cui la stessa OECD (7). Esso segue una ben precisa struttura:

```
{
  "header": {...},
  "dataSets": [...],
  "structure": {...}
}
```

La prima parte riporta informazioni sul mittente del file insieme all'URI utilizzato per reperire il file.

Nella lista "dataSets", che in questo caso ha un solo elemento, coincidente al dataset scaricato, i valori sono dizionari, uno dei quali, chiamato "series", che contiene tutte le serie, indicizzate secondo le tecniche stabilite da SDMX, con numeri in quantità pari alle dimensioni presenti nel dataset separati da ":". Ogni serie contiene la lista delle osservazioni, in questo caso non utile. Gli identificatori delle serie vengono salvati in una lista che viene iterata, poiché questi numeri coincidono uno ad uno ad un identificatore di ogni dimensione. Le dimensioni sono riportate nella sezione "structure" che possiede, tra altri valori quali il nome del dataset e annotazioni varie, un dizionario "dimensions" che contiene valori tra cui "series", un array composto dalle dimensioni che riporta per ognuna la posizione nell'identificatore delle time series, il nome della dimensione stessa e un array "values" i cui indici corrispondono al numero indicato negli identificatori sopra citati, e contenente due coppie chiave valore, "id" e "name". Ognuno di questi ultimi corrisponde ad una descrizione della dimensione, che ho utilizzato per costruire la descrizione da inserire in Elastic.

Ecco un esempio per chiarire la complessa struttura SDMX, prendendo dal dataset "TABLE5", la cui descrizione è: "Aid (ODA) by sector and donor [DAC5]", la time series con identificatore "53:1:0:1".

Possiamo risalire, dalla sezione "structure" del file ottenuto da OECD, alla descrizione di ognuna delle dimensioni "Donor", "Sector", "Aid type" ed "Amount type" a cui ognuno di questi numeri corrispondono, ovvero:

- il valore con indice 53 della prima dimensione, ovvero "Donor"
- il secondo valore della seconda lista, per la dimensione "Sector"
- il primo risultato della dimensione "Aid type"

- il secondo indice dell'ultima dimensione, relativa all'"Amount type"

Quindi, ad ognuno di questi indici corrisponderà una breve descrizione, in questo caso:

- "Donor" = UNICEF
- "Sector" = I.1. Education
- "Aid type" = Total ODA
- "Amount type" = Constant Prices

Queste descrizioni vengono quindi unite alla descrizione del dataset e inserite nel documento corrispondente alla serie "53:1:0:1".

Lo stesso procedimento viene effettuato per la costruzione dell'id della time series, utilizzando il campo "id" delle dimensioni:

- table5.963.110.528.D

Una volta costruiti *id* e *descrizione*, essi vengono inseriti nella struttura Bulk richiesta da Elastic, rispettivamente in due stringhe "create" e "doc":

- {"create": { "\_index": "timeseries", "\_type": "external",  
"\_id": "oecd.all.table5.963.110.528.d " } }
- {"id": "oecd/all/table5.963.110.528.d ", "name": "Aid (ODA) by sector and donor [DAC5]",  
"description": "UNICEF and I.1. Education and Total ODA and Constant Prices", "source": "OECD", "country": "ALL" }

Essi vengono scritti nel file "OECD.elastic" e si passa ad analizzare la time series successiva.

Ad ogni iterazione, ho inserito un controllo sulla dimensione del file, poiché di default Elastic limita la dimensione dei file *Bulk* a 104 Megabyte, quindi, se il file ha superato la dimensione di 100 Megabyte, viene lanciato il sottoprocesso "oecdIndexCreate", un piccolo *script* in *bash* che con una chiamata *curl* carica il file su Elastic (14).

A questo punto il file viene riaperto in scrittura, che in Python equivale a sovrascrivere, e si continua ad elaborare le serie, fino al completamento del dataset, evitando di caricare file troppo grandi su Elastic.

```
with open('glimpse_ts/indexer/OECD.elastic', 'w') as fp:
    for i in range(len(key_list)):
        if os.path.getsize("glimpse_ts/indexer/OECD.elastic") > 100000000:
            subprocess.call('. glimpse_ts/indexer/oecdIndexCreate.sh', shell=True)
            fp.close()
        fp = open("glimpse_ts/indexer/OECD.elastic", "w")
```

Se invece il body risulta errato, viene istanziata una variabile booleana “*timedout*” con valore *False*: essa diventa vera se l’URL di questa iterazione è stato scritto, durante l’esecuzione del metodo “*get\_body*”, nel file “*oecd\_timedout*”. Questo è utile per la scrittura del *logger*, poiché in base al valore di “*timedout*” il dataset risulta vuoto per tempo limite raggiunto piuttosto che per problemi del server.

Alla fine dell’esecuzione, il file “*oecd\_failed*” può essere utilizzato per rilanciare velocemente l’algoritmo sui dataset da cui non si è riusciti ad ottenere dati, con lo scopo di migliorare la completezza dell’indice.

L’algoritmo permette di costruire un indice per oltre 6 milioni di time series appartenenti ad OECD in circa 9 ore, contro le poco più di 30000 fornite dal nostro competitor Quandl. Data comunque la durata nell’ordine delle ore, abbiamo sviluppato uno script in *bash* che, inserendo il codice della fonte, in questo caso OECD, esegue il modulo Python ad esso collegato per la creazione dell’indice in background tenendone traccia con un *logger* esterno all’algoritmo.

Questo diminuisce la priorità del processo e quindi aumenta anche i tempi di esecuzione, ma ho potuto notare anche una diminuzione del fallimento delle singole query e degli eventuali *timeout*.

## 4.3 Indicizzatori per altre fonti

Per quanto riguarda le altre fonti di cui ho costruito un indice, la costruzione è stata decisamente più semplice.

Lo sviluppo è stato quindi eseguito in “*bash scripting*” data la bassa complessità, per avere migliori prestazioni.

Le fonti in questione sono il Dipartimento dell’Energia americano, DOE, il Fondo Monetario Internazionale, IMF e la Banca Centrale Europea, ECB. Tutti i dati sono



stati prelevati tramite Quandl, come per gli adattatori sviluppati per le suddette fonti (9).

Questo provider permette il download dell'intera lista di time series di un database, costruendo una semplice richiesta HTTP, come segue:

- [https://www.quandl.com/api/v3/databases/<database\\_name>/codes](https://www.quandl.com/api/v3/databases/<database_name>/codes)

La chiamata restituisce un file Excel che in ogni riga contiene l'identificatore della time series nel formato "FONTE/ID" e la sua descrizione, separate da una virgola, come mostrato:

1	DOE/EER_EPMRR_PE3_Y35NY_DPG,New York Harbor Reformulated RBOB Regular Gasoline Future Contract 3
2	DOE/RWTC,"WTI Crude Oil Spot Price Cushing, OK FOB"
3	DOE/EER_EPD2F_PF4_Y35NY_DPG,New York Harbor No. 2 Heating Oil Spot Price FOB
4	DOE/RBRTE,Europe Brent Crude Oil Spot Price FOB
5	DOE/EER_EPD2F_PE3_Y35NY_DPG,New York Harbor No. 2 Heating Oil Future Contract 3
6	DOE/EER_EPMRR_PF4_Y05LA_DPG,Los Angeles Reformulated RBOB Regular Gasoline Spot Price
7	DOE/EER_EPD2DXL0_PF4_RGC_DPG,U.S. Gulf Coast Ultra-Low Sulfur No 2 Diesel Spot Price
8	DOE/EER_EPMRU_PF4_Y35NY_DPG,New York Harbor Conventional Gasoline Regular Spot Price FOB
9	DOE/EER_EPIK_PF4_RGC_DPG,U.S. Gulf Coast Kerosene-Type Jet Fuel Spot Price FOB
10	DOE/EER_EPLPA_PF4_Y44MB_DPG,"Mont Belvieu, TX Propane Spot Price FOB"
11	DOE/EER_EPMRU_PF4_RGC_DPG,U.S. Gulf Coast Conventional Gasoline Regular Spot Price FOB
12	DOE/EER_EPMRR_PE1_Y35NY_DPG,New York Harbor Reformulated RBOB Regular Gasoline Future Contract 1
13	DOE/RCLC2,"Cushing, OK Crude Oil Future Contract 2"
14	DOE/RCLC3,"Cushing, OK Crude Oil Future Contract 3"
15	DOE/RCLC1,"Cushing, OK Crude Oil Future Contract 1"
16	DOE/RCLC4,"Cushing, OK Crude Oil Future Contract 4"

Figura 6 - Esempio di struttura dei file offerti da Quandl

Il file ha richiesto una migliore formattazione, effettuata tramite l'utilità di elaborazione di testo "*sed*" offerta dal *bash scripting*, e consistente:

- nella sostituzione della “,” che separa id e descrizione con un “;”, per distinguerla da eventuali virgole presenti nella seconda, che potrebbero creare problemi nella suddivisione dei due campi
- nella rimozione della source da ogni riga, in tutti e 3 i casi cancellando i primi 4 caratteri (rispettivamente “DOE/”, ”ODA/” per l’IMF ed “ECB/”), non utili per l’inserimento nell’indice e, nel secondo caso, anche fuorvianti

A questo punto viene effettuato un controllo per verificare l'esistenza del file “.elastic” della fonte, e, in caso positivo, esso viene cancellato per far spazio al nuovo indice, utile se si sta effettuando un aggiornamento ed il file non sia stato rimosso manualmente. Successivamente il file viene scorso e per ogni riga l'id e la descrizione vengono salvate in due variabili, rispettivamente “*series*” e “*title*”. Questa operazione

viene effettuata grazie al filtro “*awk*” utilizzando come separatore i “,” inseriti precedentemente.

L’id viene inoltre convertito in *lowercase*, condizione necessaria per gli identificatori in Elastic. Le due variabili vengono quindi inserite, come visto per OECD, nella coppia di stringhe “*create*” e “*doc*”. In questo caso, i campi “name” e “description” in “doc” coincideranno, mentre nel caso precedente la prima riportava il nome del dataset di appartenenza, qui non presente.

Le due stringhe vengono quindi scritte nel file “*fonte.elastic*”.

Alla fine della lettura del file ottenuto da Quandl, i documenti relativi alla fonte vengono cancellati da Elastic, e il file ottenuto dall’elaborazione viene caricato, ricreando l’indice:

```
# delete
curl -XDELETE 'http://localhost:9200/_all/_query?q=source:ECB'
# load
curl -s -XPOST localhost:9200/_bulk --data-binary @glimpse_ts/indexer/ECB.elastic;
echo
```

L’avvio della procedura viene eseguita con le stesse modalità di OECD dallo script “*create\_index*” indicando la fonte.

Il procedimento appena descritto è lo stesso per le 3 fonti, e offre tempi di esecuzione molto bassi, data la ridotta dimensione delle time series nei database: DOE possiede 11876 time series, mentre IMF ne ha 8379, ed entrambe vengono caricate in Elastic in circa 10 minuti.

L’indice per ECB, con 150,104 elementi, viene costruito ed inserito in Elastic in circa un’ora.

## Capitolo 5 – Export delle serie

In questo capitolo si parla degli algoritmi di export in diversi formati da me implementati, a partire dal formato base per le immagini, PNG, per poi trattare più ampiamente del formato SDMX, di cui si è già parlato nel capitolo precedente. Infine, l'ultimo formato è quello nativo di Glimpse, ovvero un particolare tipo di JSON, con cui mantiene i dati sul database.

### 5.1 PNG

Glimpse permette, nella sezione “*Collect*”, di vedere il grafico della time series non appena effettuato il download della stessa, cliccando l'icona che si può vedere nell'immagine sotto il grafico.

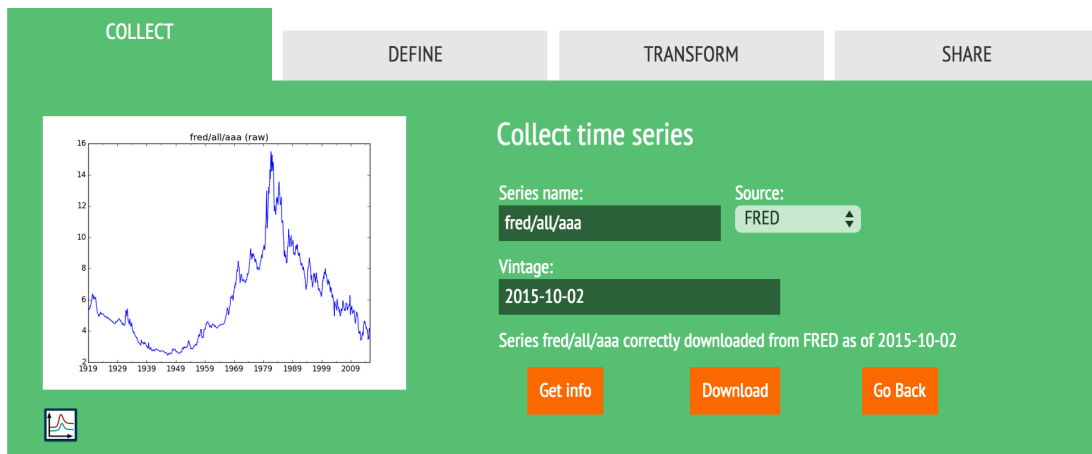


Figura 7 - Grafico visibile al posto dei metadati dopo il download di una time series

Questo per dare immediatamente all'utente un'idea della serie scaricata. Nella sezione “*Transform*” si possono applicare trasformazioni alla time series e subito vederne i risultati cliccando sull'icona, come sopra.

Per scaricare effettivamente questo grafico, bisogna utilizzare, come per tutti gli altri formati, la sezione “*Share*”. In particolare per il download della serie in formato PNG, da me sviluppata, bisogna selezionare il formato dal menù a tendina e cliccare “Export timeseries”.

Della scrittura dei file derivanti da questa operazione si occupa la classe “*Exporter*”, che possiede un metodo per ognuno dei formati.

In particolare, nel metodo “*dataset\_png*” viene inizialmente effettuato il controllo del *vintage*, delegato al modulo “*vintage.py*”. Il *vintage* viene quindi utilizzato, insieme alle informazioni sulla time series ed eventualmente alla trasformazione su essa

effettuata, per istanziare un oggetto DataFrame, definito da “*pandas*” (15), una libreria open source che offre tools per l’analisi di dati e strutture dati per Python. Il DataFrame è una di esse, ed ha una struttura bidimensionale organizzata in colonne potenzialmente di diversi tipi, adatta alla rappresentazione di time series. In particolare esso offre anche la funzione *plot*, che permette appunto la rappresentazione grafica dei valori che contiene.

L’oggetto restituito permette, con il metodo “*get\_figure*”, la trasformazione della rappresentazione in un’immagine, che viene quindi scaricata per l’utente in un file PNG.

L’export è orientato alle singole serie, e non prevede un export per i dataset.

## 5.2 SDMX adattato a Glimpse

Come abbiamo visto nell’analisi dei file SDMX forniti da OECD, questo formato organizza le serie suddividendole in base alle dimensioni e alle osservazioni. Questa struttura ovviamente non è possibile per l’export da Glimpse, che mantiene traccia delle dimensioni nella descrizione, ma solo delle serie raccolte da questa fonte.

Si è quindi optato per un’organizzazione differente per permettere l’export di serie singole ma anche di dataset. Il file conserva ovviamente la suddivisione iniziale dell’SDMX:

- *header*, che riporta informazioni sulla fonte
- *dataSets*, contenete le serie con le relative informazioni
- *structure*, che descrive la corrispondenza degli identificatori in *dataSets*

Nella prima parte ho inserito un identificatore generato dal modulo Python “*uuid*”, la data di creazione del file e informazioni sul “*sender*”, che in questo caso è Glimpse.

Nella sezione “*dataSets*” si trova il dizionario “*series*” utilizzato dall’SDMX, che si compone di coppie chiave valore corrispondenti alle singole osservazioni delle serie. Come abbiamo visto precedentemente, le chiavi consistono in serie di numeri divisi da “.”, in questo caso i numeri sono due, il primo corrispondente alla time series, e il secondo all’osservazione.

È molto probabile che le serie non abbiano tempi di inizio e fine coincidente e stessa frequenza, quindi ho dovuto identificare la prima e ultima osservazione del dataset. Per farlo, ho inizialmente definito due variabili “*datetime*”, “*obs\_start*” ed “*obs\_end*”,

la prima con la data odierna, e l'altra con una data molto remota. A questo punto, ho scorso la lista delle serie presenti nel dataset caricandole da Mongo, e, per ognuna, ho salvato la lista delle osservazioni in una lista *“observations\_list”* istanziata insieme alle due variabili temporali, salvato temporaneamente *“observation\_start”* ed *“observation\_end”* per confrontarle rispettivamente con *“obs\_start”* ed *“obs\_end”* e rimpiazzando il valore di quest'ultime se la prima risulta più piccola o la seconda più grande. Queste informazioni nei metadati sono salvate come stringhe nel formato *“YYYY-MM-DD”*, quindi prima del confronto ho rimosso i *“-”* separatori e le ho tradotte in date. Infine ho salvato, per ogni time series, anche un dizionario contenente id e titolo, per distinguerle successivamente, e le date delle osservazioni in una lista.

```
observations_list = []
ts_values = []
obs_lists = []
# Random dates initialized to find the very first and last dates in the ds
obs_start = datetime.datetime.now()
obs_end = datetime.datetime.strptime('19000101', '%Y%m%d')

for ts_name, tr in ts_to_trans.items():
    ts = TimeSeries(str(ts_name))
    ts.load()
    # list of obs_lists, length is the number of timeseries in the dataset
    obs_lists.append(ts.obs_list)
    ts_start = datetime.datetime.strptime(ts.metadata["observation_start"].replace("-",
    ", ''"), "%Y%m%d")
    if ts_start < obs_start:
        obs_start = ts_start
    ts_end = datetime.datetime.strptime(ts.metadata["observation_end"].replace("-",
    ', '''), "%Y%m%d")
    if ts_end > obs_end:
        obs_end = ts_end
    ts_values.append({'id': ts.metadata['id'], 'name': ts.metadata['title']})
    for val in ts.obs_list:
        observations_list.append(val["date"])
```

Alla fine del ciclo, ho trasformato in *set* questa lista, con il fine di eliminare eventuali duplicati, e la ho riconvertita in lista per poterla ordinare, cosa non possibile per i *set*. Il *sort* della lista è stato effettuato tramite una funzione *“lambda”* di Python da me definita. A questo punto le date sono state salvate in una lista di dizionari seguendo lo schema di SDMX, con due chiavi, *“id”* e *“name”*.

Effettuata quest'ultima operazione, si passa all'indicizzazione delle osservazioni secondo la struttura definita, al fine di inserirle nella sezione *“dataSets”*. Viene istanziata una lista *“data\_dict”* che conterrà le osservazioni e si entra in tre cicli annidati scorrendo, nell'ordine, la lista delle date *“final\_obs\_list”*, la lista delle liste delle osservazioni *“obs\_lists”* delle singole serie, e ognuna di queste liste: quando si identifica una data a cui è collegata una delle osservazioni delle serie, ci si trova davanti a due opzioni:

- se il valore dell’osservazione non è nullo, a “*data\_dict*” viene aggiunta la coppia chiave valore nel formato “*i:j*” : “*float*”, dove *i* corrisponde al numero della time series e *j* all’identificatore della data, mentre *float* è il valore dell’osservazione *x* presa dalla lista di osservazioni della serie *i*
- se il valore è nullo, alla lista viene aggiunta la stessa struttura, ma il valore viene rimpiazzato dalla stringa “*null*”

```
# Creating a SDMX-JSON formatted observations indexed by observation lists
for j in range(len(final_obs_list)):
    for i in range(len(obs_lists)):
        for x in range(len(obs_lists[i])):
            if final_obs_list[j]['id'] == obs_lists[i][x]['date']:
                if obs_lists[i][x]['value'] == '.' or obs_lists[i][x]['value'] ==
'null':
                    data_dict.append({str(i)+":"+str(j): ["null"]})
                else:
                    data_dict.append({str(i)+":"+str(j):
[float(obs_lists[i][x]['value'])]})
```

A questo punto la lista viene inserita in un dizionario “*serie\_values*” per rispettare la struttura dell’SDMX.

Il dizionario viene quindi trasformato in un “*OrderedDict*”, ovvero un dizionario ordinato secondo una chiave da me definita per effettuare il *sort* per il primo valore dell’identificatore, coincidente alla time series, e successivamente in base al secondo, che rappresenta gli *id* delle osservazioni. Dopo l’ordinamento, il dizionario viene assegnato al campo “*dataSets*” del file.

Si passa quindi alla costruzione dell’ultima sezione del file SDMX, “*structure*”, in cui si trovano le chiavi per conoscere l’appartenenza delle osservazioni. Essa possiede il dizionario “*dimensions*”, in cui vengono inseriti identificatore e titolo delle time series, mentre nella lista “*observations*” viene riportata la lista delle osservazioni estratta al momento del caricamento da Mongo da ogni serie.

Infine, le tre sezioni vengono inserite in un dizionario radice, e viene effettuato il parse in JSON dell’oggetto. Il risultato viene quindi restituito e scaricato sul PC dell’utente.

## 5.3 JMO, il formato di storage

Come visto nel capitolo riguardante gli adattatori, le serie storiche vengono conservate sul database Mongo in un particolare formato, composto da un identificatore, un array “*data*” per le osservazioni e un dizionario “*metadata*” che riporta le informazioni caratterizzanti.















 <b>_id</b>	ObjectId("55fee6605d9c5209d90d6b37")	ObjectId
▶  <b>data</b>	Array [2]	Array
▼  <b>metadata</b>	{ 13 fields }	Object
 <b>last_updated</b>	2015-09-20	String
 <b>observation_start</b>	2011-01-01	String
 <b>title</b>	VII - TEC by ownership (domestic or foreign) and Tr...	String
 <b>seasonal_adjustment</b>	Not seasonally adjusted	String
 <b>observation_end</b>	2012-01-01	String
 <b>realtime_end</b>	2015-09-20	String
 <b>discontinued</b>	No	String
 <b>source</b>	OECD	String
 <b>frequency</b>	Annual	String
 <b>units</b>	No unit	String
 <b>source_name</b>	Organisation for Economic Co-operation and Devel...	String
 <b>realtime_start</b>	2015-09-20	String
 <b>id</b>	oecd/all/tec7_rev4.2.lux.2.e_eu.unk.unspe	String

Figura 8 - Esempio di file su Mongo mostrato tramite Robomongo

Per l'export in questo formato, viene effettuato inizialmente il controllo sul *vintage*, con il quale poi la serie viene caricata dal database. A questo punto, viene creato il *pandas* utilizzando il vintage convertito e la trasformazione selezionata. Questo viene effettuato per avere la possibilità di reperire i dati in base al vintage, nel caso di time series che lo permettano, utilizzando il metodo *"setDataFromPandaTsByVintageAndTransformation"* dell'oggetto *"TimeSeries"*.

A questo punto, viene istanziato un dizionario *obj*, a cui vengono assegnate due campi: *"data"* e *"metadata"*. Ovviamente la prima viene riempita con la lista delle osservazioni presa dalla time series, e la seconda con i metadati.

Dopo aver effettuato il *dump* dell'oggetto in JSON, esso viene scaricato in un file formattato per l'utente.

## Capitolo 6 – Usabilità

In questo capitolo si parla dell’inserimento della documentazione che fornisce indicazioni su come utilizzare Glimpse insieme a informazioni per gli sviluppatori che desiderano interfacciarsi e delle modifiche grafiche per migliorare la fruizione e la consultazione dei dati all’utente.

### 6.1 La documentazione di Glimpse

Dopo lo sviluppo degli *adapter* e delle API REST che definiscono un’interfaccia a Glimpse per gli sviluppatori, si è resa necessaria l’introduzione di una documentazione curata per l’utilizzo dell’applicazione. È per questo che ho implementato la sezione *Docs* di Glimpse. Essa è accessibile cliccando l’icona di un documento nella parte inferiore della pagina principale.



Figura 9 - Particolare del footer del portale web

Essa porta alla sezione “/docs” del sito, e in particolare alla pagina principale di Glimpse Docs, che eredita lo stile di Glimpse, con le 4 etichette per navigare. In questo caso le etichette presenti sono: “Home”, “Sources”, “API” e “Back to Glimpse”.



Figura 10 - Header della documentazione di Glimpse

Nella prima sezione, come si può intuire dal titolo, viene spiegato cosa sia Glimpse, in particolare la sua utilità nel riunire le fonti in una semplice interfaccia, e i suoi valori di trasparenza, consapevolezza dell’importanza di fornire dati di qualità e affidabilità. Vengono infine citate alcune delle tecnologie e delle tecniche utilizzate. Questa sezione è stata ottenuta con semplici tag HTML, non necessitando di un particolare livello di interazione con l’utente. Inoltre risulta gradevole alla vista e non sforza gli occhi utilizzando colori non contrastanti.

Nella seconda sezione, “Source”, si trova la lista delle fonti a cui ci siamo interfacciati, e per ognuna una descrizione di utilizzo. Nella parte superiore si trova la lista delle



fonti, e ad ogni nome coincide un link che porta nella pagina alla spiegazione della *source* di interesse. Questo risultato è ottenuto tramite le *àncore*, una funzionalità dell'HTML.

Un'ancora, o link interno, è un indice interno al documento, dotato di un nome:

- `<a name="FRED"></a>`

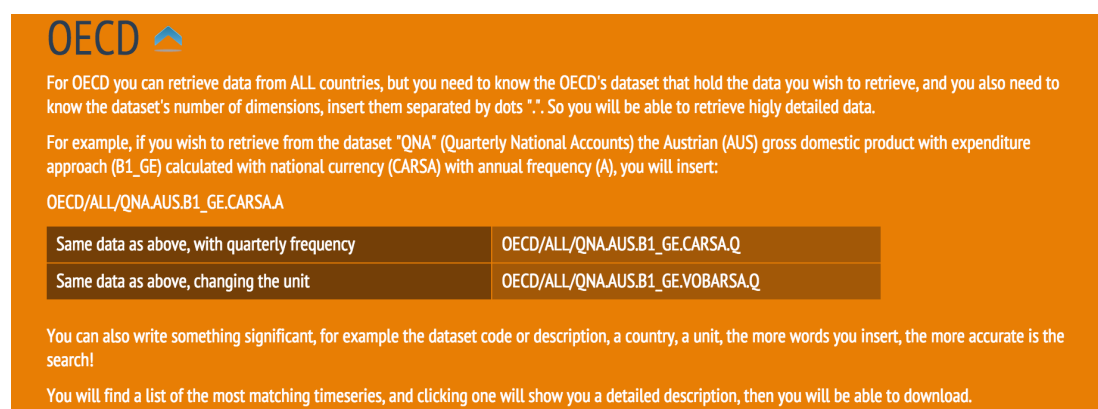
Si può notare la mancanza dell'attributo che indica il collegamento (*href*), infatti esse non vengono viste come link, ma la loro formattazione è indistinguibile dal “normale” testo. In un ipotetico indice è allora possibile far riferimento all'ancora presente all'interno del documento attraverso un link che punti ad essa:

Quindi ogni fonte è raggiungibile più velocemente da questa lista nella parte superiore della pagina.



*Figura 11 - Particolare della lista delle fonti nella documentazione*

Una volta cliccato il link alla fonte che interessa, viene spostata la visuale sulla sua descrizione, che varia in base alla complessità del funzionamento. Per ognuna vengono inoltre riportati uno o più esempi di input possibili con una descrizione, per mostrare all'utente l'utilizzo e facilitargli la ricerca. Accanto ad ogni fonte ho inserito una freccia che riporta in cima alla pagina, per velocizzare la fruizione della documentazione. Questo è stato realizzato sempre con le *àncore*, definendone una chiamata “Source” in cima alla pagina e linkandola con un “*href="#Sources"*” che contiene la freccia.



*Figura 12 - Esempio di documentazione per la fonte OECD*

La sezione seguente, “*APP*”, è destinata alla documentazione dell’interfaccia REST di Glimpse, di cui non mi sono occupato durante il tirocinio.

Infine, l’ultima sezione è in realtà un semplice link che, come si può intuire, riporta l’utente alla pagina principale del sito dopo che questo ha appreso le basi dell’utilizzo di Glimpse ed è pronto ad fruire dei servizi che offre.

## 6.2 Miglioramento della resa grafica

Durante lo sviluppo, abbiamo incontrato problemi grafici che potevano impedire un utilizzo ottimale e piacevole dell’applicazione. Per questo via via è sorta la necessità di intervenire sull’estetica o su problemi visivi.

Ad esempio, il primo dei compiti da me portati a termine durante il tirocinio è stata la limitazione della lunghezza dei campi dei metadati: questo perché ad esempio, in caso di descrizioni delle serie molto lunghe, andavano a compromettere la forma della tabella che li riporta.



Timeseries name	oecd/all/qna.aus.b1_ge.carsa.q
Description	Quarterly National Accounts and Australia and G...
Source	OECD

*Figura 13 - Particolare dei metadati con descrizione troncata*

Per questo ho applicato un filtro fornito dal framework *Django*, “*truncatechars*”, che limita la lunghezza delle variabili iniettate nell’HTML ad un numero di caratteri inserito dopo il filtro:

- `{{ ts.metadata.title|truncatechars:50 }}`

Esso misura la lunghezza della stringa in output e, se superiore al valore numerico che riporta, sostituisce i caratteri seguenti ad esso con dei “...”.

In questo modo la tabella non è sproporzionata, ma l’utente non può leggere la descrizione completa della serie.

Perciò ho inserito una funziona *JavaScript* che si attiva al passaggio del mouse sui campi dell’id e della descrizione, gli unici che in alcuni casi superavano le limitazioni imposte dal filtro, e che rimuove il filtro mostrando la descrizione (o l’id) completa all’utente.

Timeseries name	oecd/all/qna.aus.b1_ge.carsa.q
Description	Quarterly National Accounts and Australia and Gross domestic product - expenditure approach and National currency, current prices, annual levels, seasonally adjusted and Quarterly
Source	OECD
First date	1960-01-01
Last date	2015-02-01
Frequency	Quarterly
Seasonal adjustment	Seasonally adjusted
Updated on	2015-10-04
Discontinued	No

Figura 14 - Metadati con descrizione completa


In seguito ho inserito le icone per login tramite Gmail, Linkedin ed Amazon nella parte destra superiore, in linea con l'*header* della pagina, e le ho ridimensionate per via della diversa dimensione.


GLIMPSE




Figura 15 - Particolare dell'header di Glimpse

Infine, ho anche curato l'output delle ricerche per descrizione, che va a sostituire la tabella dei metadati con una lista di time series derivanti dalla ricerca effettuata dall'utente. Per ogni serie viene mostrata l'icona della fonte, e poiché la fonte è data esse saranno tutte uguali, affiancata dalla sigla della stessa, mentre sotto di esse è riportato l'identificatore, cliccabile in modo da effettuare immediatamente la ricerca per id, e infine la descrizione completa. L'icona viene inserita in ogni div risultante dalla ricerca tramite JavaScript utilizzando un URI che punta alla directory in cui si trovano i loghi completato dalla source che è stata scelta dall'utente.

 DOE  
doe/all/eer\_epmrr\_pe1\_y35ny\_dpg : New York Harbor Reformulated RBOB Regular Gasoline Future Contract 1

 DOE  
doe/all/eer\_epd2f\_pe4\_y35ny\_dpg : New York Harbor No. 2 Heating Oil Future Contract 4

 DOE

### Collect time series

Series name:  Source:

Choose the series

[Get info](#)

Figura 16 - Esempio di ricerca per descrizione

Una volta selezionata la serie, appariranno i metadati come al solito, ma ci sarà accanto a “*Download*” il pulsante “*Go back*” che permette di tornare alla lista delle serie.

## Capitolo 7 – Conclusioni

L'obiettivo del tirocinio è consistito nell'ampliamento incrementale del provider Glimpse.

Esso può considerarsi raggiunto, poiché lo stesso framework di sviluppo ammette l'iniziale difficoltà di approccio, garantendo però un graduale adattamento con conseguente aumento della produttività, come abbiamo potuto vedere nel capitolo 2 dagli Sprint Burndown Chart.

Abbiamo infatti migliorato pian piano i nostri metodi di approccio, diminuendo i tempi e riuscendo a portare a termine i compiti degli Sprint in minor tempo e in maniera migliore, andando quindi ad intervenire su quelli precedenti con opere di *refactoring* ove necessario, su argomenti o problemi che inizialmente non avevamo preso in considerazione o non avevamo notato. Inoltre il tirocinio ha aggiunto tra le mie competenze la conoscenza del linguaggio Python, e alcune basi di *bash scripting* e *JavaScript*, linguaggio che spero di approfondire. Infine, ma non per ultimo, mi ha dato una prospettiva migliore di lavoro in gruppo, seppur non del tutto in ambito lavorativo, poiché il tirocinio è stato svolto all'interno dell'Università degli studi di RomaTre, in particolare nel laboratorio Basi di Dati, ed in collaborazione con il mio collega e amico Marco Faretra, supportati dall'Ing. Luigi Bellomarini.

L'applicazione all'inizio di questo periodo offriva la ricerca per id per una sola fonte. Il numero di quest'ultime è stato portato a 9, sono aumentati anche i formati disponibili per l'esportazione dei dati, ed è stata aggiunta una buona documentazione sulle funzionalità e sull'utilizzo del servizio. A mio parere però, la modifica fondamentale e più utile è stata la creazione degli indici su Elastic Search, che ha reso possibile la ricerca su tutte le serie di ogni singola fonte, supportata da un autocompletamento aggiornato ad ogni carattere inserito dall'utente. Questo apre la strada alla scalabilità offerta da Elastic e da MongoDB, permettendo a Glimpse di offrire potenzialmente una banca dati enorme, che come detto si costruisce in proporzione alle ricerche effettuate dall'utenza. Glimpse si propone a tutti gli effetti come competitor di altri provider di dati. Questo filone è particolarmente sviluppato negli Stati Uniti, ma ancora in via di sviluppo in Italia e in Europa.

## Bibliografia e sitografia

Bellomarini L. Laurenza E., Glimpse - The open provider of financial time-series; 2015.

MongoDB, MongoDB for GIANT Ideas | MongoDB, MongoDB

<https://www.mongodb.org/>

(Ultima consultazione 21/09/2015)

Mongo, Capitolo 1: Introduction to MongoDB In: MongoDB, MongoDB Documentation; 29/06/2015

Zachary Tong, Get Started with Elasticsearch, Elastic Search

<https://www.elastic.co/webinars/get-started-with-elasticsearch>

(Ultima consultazione 21/09/2015)

Elastic, Basic Concepts, Elasticsearch Reference

[https://www.elastic.co/guide/en/elasticsearch/reference/current/basic\\_concepts.html](https://www.elastic.co/guide/en/elasticsearch/reference/current/basic_concepts.html)

(Ultima consultazione 30/09/2015)

Deemer P. Benefield G. Larman C. Vodde B., The Scrum primer - A Lightweight Guide to the Theory and Practice of Scrum. Version 2.0; 2012

OECD Data, OECD data for developers, OECD Documentation

<https://data.oecd.org/api/>

(Ultima consultazione 22/09/2015)

Quandl, Quandl Financial and Economic Data, Quandl

<https://www.quandl.com/>

(Ultima consultazione 21/09/2015)

Quandl, API Documentation, Quandl Documentation

<https://www.quandl.com/docs/api>

(Ultima consultazione 21/09/2015)

UNData, UNdata | api manual, UN Documentation

<http://data.un.org/Host.aspx?Content=API>

(Ultima consultazione 02/09/2015)

UNData, NSI .NET Client, SDMX Browser

<http://data.un.org/SdmxBrowser/start>

(Ultima consultazione 02/09/2015)

Selenium, Selenium - Web Browser Automation, Selenium

<http://www.seleniumhq.org/>

(Ultima consultazione 02/09/2015)

Python Software Foundation, Quick start — Whoosh 2.5.6 documentation, Whoosh Documentation

<https://pythonhosted.org/Whoosh/quickstart.html>

(Ultima consultazione 02/10/2015)

Elastic, Bulk API, Elastic Search Bulk API Documentation

<https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html>

(Ultima consultazione 02/10/2015)

Python Software Foundation, Python Data Analysis Library — pandas, Pandas Library

<http://pandas.pydata.org/>

(Ultima consultazione 02/10/2015)