Dipartimento di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica

Tesi di Laurea

Integrazione di basi di dati eterogenee per la gestione di serie storiche

Laureando

Marco Faretra

Relatore:

Prof. Luca Cabibbo

Correlatore:

Ing. Luigi Bellomarini

Anno Accademico 2014/2015

A mio Padre

Ringraziamenti

Ringrazio innanzitutto il Professor Luca Cabibbo, Relatore e l'Ing. Luigi Bellomarini, Correlatore, per i suggerimenti e il supporto ricevuto durante questo periodo.

Una delle cose più importanti che ho appreso durante il corso di questi tre anni è non porsi mai dei limiti nella propria vita e non arrendersi al primo ostacolo.
Inoltre ho imparato quanto è importante essere costanti nello studio per poi ottenere i risultati voluti.
In questi tre anni ho trovato delle persone speciali che tengo a ringraziare con tutto il cuore.

Un ringraziamento lo dedico a tutte quelle persone con cui ho condiviso piacevoli risate come Daniele Giunta, Antonio Matinata, Andrea Salvoni, Dalila Rosati.

Un ringraziamento lo dedico agli amici di sempre come Davide Potenza, Francesco Lorenzato, Simone Grazioli, Davide Giusti.

Ringrazio il gruppo di amici con cui ho trascorso più tempo insieme e con le quali passo ogni giornata come Antonio Martinelli, Alessio Zoccoli, Simone Rocchi, Lorenzo Goldoni.

Tengo a fare un ringraziamento particolare al mio migliore amico Federico Ginosa che ha sempre creduto in me sin dall'inizio, forse più di quanto non lo faccia io.

Dedico un ringraziamento speciale alla mia ragazza Claudia Romeo, che mi ha sopportato durante questi 2 anni e soprattutto negli ultimi mesi, a cui devo tutto quello che ho.

Infine tengo a ringraziare tutta la mia famiglia che mi ha sostenuto, in particolare ringrazio mio fratello Fabio Faretra, mia sorella Gaia Stazi, mia madre Carla Mancini e Flavio Stazi, che hanno reso possibile tutto ciò.

Vi voglio bene, scusate se ho dimenticato qualcuno.

Introduzione

Glimpse è una piattaforma per l'integrazione di dati provenienti da fonti diverse ed eterogenee. In particolare, Glimpse attualmente tratta serie storiche (*time series*) di tipo economico, statistico e finanziario.
Le serie storiche sono funzioni reali di una o più variabili in dipendenza dal tempo.
In economia, nella statistica e nella finanza si utilizzano per modellare varie grandezze di interesse, ad esempio: il prezzo del petrolio giorno per giorno, il prodotto interno lordo di un paese per anno e così via.

Diverse categorie di utenti fanno uso delle serie storiche per il loro business quotidiano. Economisti e statistici, in particolare hanno l'esigenza di reperire serie da una molteplicità di fonti diverse per la loro attività di analisi e ricerca. Queste operazioni sono tipicamente effettuate attraverso una ricerca manuale nelle interfacce grafiche fornite dalle sorgenti.
Ciò rappresenta un problema perché l'utente ha molte piattaforme con cui interagire, spesso molto complicate, quindi deve necessariamente conoscere tutti i formati distribuiti dalle fonti per lavorare con i dati di interesse.
Glimpse risolve questo problema per facilitare il lavoro dell'utente, fornendo un'unica piattaforma con cui interagire.
Ad esempio se un utente è interessato alla serie storica GDP ("*Gross Domestic Product*"), è necessario inserire il nome, selezionare la sorgente di interesse e l'applicazione fornisce direttamente i dati desiderati dalla sorgente richiesta.

Le serie trattate da Glimpse, derivano da diverse sorgenti: banche centrali, banche commerciali, istituti di statistica nazionali, organizzazioni sovranazionali e data providers.

Glimpse oltre ad offrire un unico strumento di estrapolazione di dati finanziari, una volta estratti i dati fornisce altre tre features di particolare interesse:
1. Effettuare delle trasformazioni matematiche, utili agli addetti ai lavori per effettuare previsioni e statistiche.
2. Esportare le stesse serie storiche in diversi formati: PNG, XLSX, JSON, ecc...

Permettendo così la condivisione dei dati in modo semplice e nei formati descritti. 3. Creare dei dataset personali effettuandone l'esportazione. Un dataset è una collezione di dati, rappresenta l'unione di una o più time series, concettualmente molto simili ad una tabella di un database.

L'obiettivo del tirocinio è stato l'ampliamento di Glimpse. Inizialmente all'interno di Glimpse sono state affrontate diverse problematiche. Le poche sorgenti disponibili da dove estrarre le serie storiche, solo FRED ("*Federal Reserve Economic Data*"), mentre l'applicazione deve fornire il maggior numero di sorgenti dal quale estrarre le serie. L'assenza di un servizio di ricerca per descrizione, per esempio stile Google, che permette di ricercare le serie storiche, un aspetto molto importante, in quanto facilita l'utente nel trovare la serie desiderata.
L'assenza di servizi base forniti all'utente, come per esempio l'autenticazione e l'autorizzazione, poiché verrà monetizzata in futuro, fornendo delle sorgenti gratuite e delle sorgenti a pagamento.
Quindi l'ampliamento si suddivide in due operazioni principali. La prima è integrare all'interno del progetto diverse sorgenti dal quale estrarre le serie storiche. Per effettuare questo servizio ho progettato un sistema che prende i dati dalla fonte e li salva sul database.
La seconda è effettuare la ricerca per descrizione, ovvero trovare le *time series* associate alla parola ricerca rispetto alla descrizione della serie, e mostrarne i risultati. Per effettuare questo servizio è necessario introdurre un indice che permette di effettuare la ricerca, che non è altro che uno storage dove inserire i dati.

L'attività di tirocinio è stata svolta presso il Laboratorio Basi di Dati, dell'Università degli studi Roma Tre all'interno di un team, composto da Marco Faretra (l'autore del presente elaborato) ed Antonio Martinelli.

Durante l'attività di tirocinio mi sono occupato dell'analisi e della progettazione per le seguenti operazioni:

1. Estrazione dei dati attraverso molte sorgenti: World Bank, Eurostat, DOE (Department of Energy), OPEC (Organization of the Petroleum Exporting Countries), BIS (Bank for international settlements).

2. Creazione degli indici per le seguenti sorgenti: World Bank, Eurostat, OPEC, BIS.
3. Integrazione dell'autenticazione attraverso i social all'interno dell'applicazione: Google, Linkedin, Amazon.
4. La progettazione di un sistema che fornisce i dati di glimpse attraverso un'interfaccia REST, che permette di trasmettere dati su http. Possibile utilizzo da futuri sviluppatori che vorranno utilizzare i dati forniti da Glimpse.
5. L'integrazione di un sistema di autorizzazione, che verifica ogni volta che l'utente effettua un'esportazione di una serie, se la sorgente è gratuita o meno e se l'utente ha un abbonamento, e in caso positivo procede all'esportazione.

La tesi è organizzata come segue:

Il Capitolo 1 tratta i requisiti e l'analisi dell'applicazione, in particolare la comprensione del problema da risolvere.

Il Capitolo 2 tratta della metodologia di lavoro utilizzata per lo sviluppo dell'applicazione, cioè il framework Scrum, dei requisiti di sistema, ovvero le tecnologie utilizzate all'interno del progetto, tra le quali MongoDB, SQLite, ElasticSearch, OAuth2.0 e dei formati utilizzati all'interno del progetto.

Il Capitolo 3 tratta la progettazione dell'estrazione e la persistenza dei dati attraverso la sorgente World Bank.

Il Capitolo 4 tratta dell'estrazione e la persistenza dei dati dalle altre sorgenti, il modo di procedere è simile a quello utilizzato per World Bank, ma con il cambiamento di alcuni dettagli.

Il Capitolo 5 descrive il motore di ricerca full-text sviluppato per serie storiche per la data dashboard dell'applicazione.

Il Capitolo 6 tratta dell'autenticazione attraverso i social integrati all'interno dell'applicazione, e dell'autorizzazione.

Il Capitolo 7 tratta della realizzazione di un'interfaccia REST per l'esportazione dei dati, tale interfaccia è composta da diversi vincoli e parametri da poter utilizzare.

1. Requisiti

Il Capitolo 1 tratta i requisiti e l'analisi dell'applicazione, in particolare la comprensione del problema da risolvere.

1.1 Estrazione dati da fonti

Uno dei principali obiettivi è integrare all'interno dell'applicazione le sorgenti da dove acquisire i dati.

Glimpse gestisce *time series* e dataset.

Le *time series* sono caratterizzate da delle proprietà, le osservazioni, dove per ogni data è associato un valore e delle informazioni che descrivono la serie; nel sistema queste proprietà sono chiamate dati e metadati.

I metadati sono composti da una serie di campi per descrivere la serie storica:

Le serie fornite dalle diverse sorgenti vengono aggiornate periodicamente, un informazione utile è quindi sapere qual è la data dell'ultimo aggiornamento della serie.

Essendo una funzione al variare del tempo, sicuramente è composta da un valore iniziale ed un valore finale alle quali è associata una data, è utile sapere la data in cui la serie inizia ad avere valore e la data in cui la serie ha l'ultimo valore.

Le serie vengono identificate attraverso una chiave unica, però è utile sapere il nome completo, ad esempio alla serie che ha come chiave GDP è associato il nome *Gross Domestic Product*, quindi risulta importante conoscere sia l'id che il nome.

Ogni serie viene estratta da una sola sorgente, quindi come descritto prima per l'id e il nome, risulta importante conoscere anche l'id e il nome della sorgente.

Una serie è caratterizzata da diversi comportamenti, uno di questi è la stagionalità, la sua particolarità è che la serie ha dei picchi durante un periodo temporale preciso.

È possibile applicare un metodo che permette di rimuovere le instabilità di carattere stagionale, che viene chiamato Destagionalizzazione. Per l'utente è importante sapere se una serie è destagionalizzata.

Le osservazioni di una serie rappresentano un valore per ogni data, possono avere una cadenza specifica, che viene descritta dalla frequenza, ad esempio se ho una frequenza

annuale, avrò un valore per ogni anno. Solitamente le frequenze utilizzate sono annuale, quadrimestrale, mensile e giornaliera.

I valori di ogni osservazione vengono riportati solamente come numeri, ad ogni serie viene quindi associata un'unità di misura che descrive appunto il significato dei valori.

Una serie come le funzioni matematiche può avere delle discontinuità, se durante le osservazioni ci sono dei valori nulli.

I dati sono composti da una lista di elementi che contiene tutte le osservazioni, ogni osservazione è rappresentata da diversi campi:

Una data ed un valore, che rappresentano rispettivamente la data dell'osservazione corrente, ed il valore associato.

Glimpse permette di effettuare le trasformazioni, è quindi presente un campo che descrive che tipo di trasformazione è stata applicata alla serie storica.

Un'altra proprietà riguardante la *time series* è il vintage, che rappresenta una data, ogni volta che eseguo l'estrazione di una serie è possibile inserire un vintage, la particolarità è che due serie storiche uguali ma con vintage diverso, possono avere delle osservazioni in comune che hanno valori diversi.

Quando un utente entra nel sistema può effettuare l'operazione di sistema di estrazione di una *time series*.

In questa operazione, l'utente effettua una ricerca della serie, inserendo l'id della serie, la sorgente e il vintage, se la serie è presente, allora il sistema mostra i metadati, altrimenti mostra un messaggio che indica che non esiste.

Se la time series è stata trovata, l'utente clicca sul pulsante "Download", allora il sistema effettua l'estrazione di dati e di metadati e li memorizza.

Una time series è identificata da una chiave univoca, in molti casi invece hanno chiave univoca la combinazione di un id con un paese.

Per esempio se ho una serie che ha come chiave univoca GDP, questa assume valori diversi se viene richiesto "Italia" o "Brasile" come paese. È stata introdotta quindi una forma standard per rappresentare le *time series*, in particolare la chiave univoca sarà rappresentata dalla coppia "paese/nome serie", che nel caso in cui una serie sia univoca

senza paese sarà nella forma “all/nome serie”. Quindi chiamiamo “chiave_univoca”, le serie storiche che non dipendono dal paese, “chiave_paese” viceversa.

L’utilizzo del sistema viene descritto in modo dettagliato attraverso il seguente caso d’uso:

Caso d’uso UC1: Ricerca per chiave

1. L’utente inserisce una stringa del tipo “paese/nome serie” oppure “nome serie” e sceglie la sorgente S.

2a. Se S è di tipo “chiave_univoca”, allora Glimpse costruisce una query utilizzando solo “nome serie”, indipendentemente dal fatto che l’utente ha specificato “paese/nome serie” o “nome serie”, eventualmente scarta paese.

2b. Se S è di tipo “chiave_paese”, allora:

2ba. Se l’utente ha inserito una stringa di tipo “paese/nome serie", Glimpse usa “paese” e “nome serie”, nei campi opportuni della URL.

2bc. Se l’utente ha inserito una stringa di tipo “nome serie”, allora è interpretata sicuramente come ricerca per descrizione.

3a. Se la ricerca per chiave dà una serie risultato, questa viene visualizzata e preparata per il download.

3b. Se la ricerca per chiave non dà alcun risultato, allora vai a “Caso d’uso UC2: Ricerca per descrizione” con descrizione = “nome serie”.

L’operazione di estrazione dei dati viene effettuata per diverse sorgenti, essendo i concetti tutti molto simili tra loro, risulta conveniente utilizzare una gerarchia di generalizzazione, in cui la superclasse “Timeseries Importer” contiene le informazioni in comune che hanno tutte le classi concettuali.

In Figura 1 troviamo una parte del modello di dominio che rappresenta la generalizzazione:

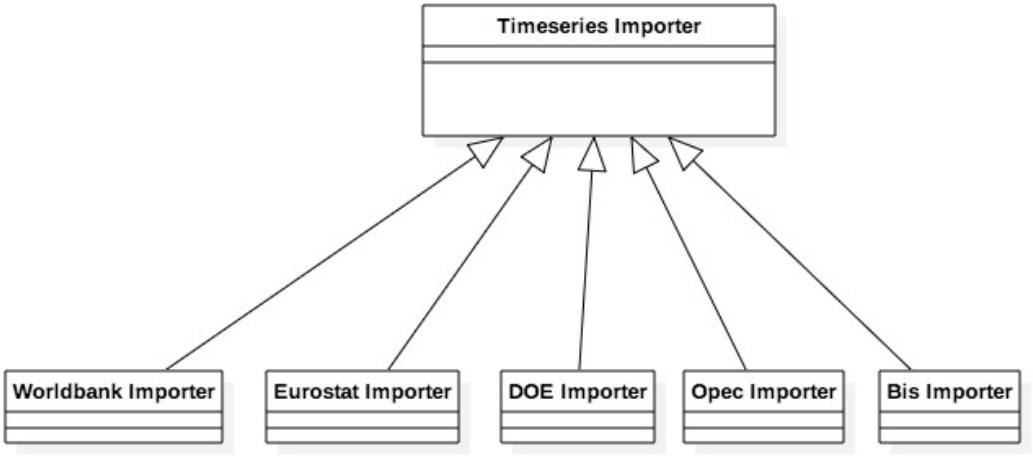

Figura 1. Parte del modello di dominio che rappresenta la generalizzazione dell'importatore

1.2 Ricerca dati da fonti

Un altro dei principali obiettivi è integrare all'interno dell'applicazione la possibilità di effettuare ricerche per descrizione.

Per poter integrare la ricerca è necessario utilizzare un sistema esterno per memorizzare tutte le serie storiche, in particolare per ogni serie vengono memorizzate soltanto le informazioni di base per identificarla e descriverla.

Per ogni serie viene memorizzato l'id e la sorgente che hanno la funzione di identificare la serie e la descrizione che permette la ricerca.

Deve essere possibile effettuare delle query a questo sistema esterno che permettono di estrarre le serie storiche desiderate.

Quando un utente entra nel sistema può effettuare l'operazione di sistema ricerca per descrizione.

L'utente inserisce una stringa da ricercare e seleziona una sorgente dalla quale vuole effettuare una ricerca.

Il sistema effettua una ricerca per prelevare i dati dal sistema esterno, mostrando all'utente una lista dei migliori 10 risultati in un menù a tendina, dove per ogni occorrenza è presente il nome della serie, una breve descrizione, ed un logo che rappresenta la sorgente dal quale si è effettuata la ricerca.

Successivamente l'utente può selezionare una serie di suo interesse e cliccarci, a quel punto viene portato alla schermata descritta precedentemente con i metadati, dove può effettuare il download, o altrimenti tornare indietro.

L'utilizzo del sistema viene descritto in modo dettagliato attraverso il seguente caso d'uso:

Caso d'uso UC2: Ricerca per descrizione:

1. L'utente inserisce una descrizione da ricercare e indica una sorgente.
2. Il sistema effettua la query e visualizza l'elenco delle serie risultato, in un menù a tendina.
3. L'utente seleziona una serie scaricata cliccando sul nome della serie.
4. Il sistema mostra al posto dell'elenco delle serie la tabella con i metadati della serie selezionata.

5a. L'utente ha letto tutto i metadati della serie e clicca su "Torna Indietro".

6a. Il sistema mostra nuovamente l'elenco della ricerca fatta precedentemente.

5b. L'utente clicca su "Download".

6b. Lo scaricamento avviene e ritorna la schermata con i nomi e i metadati.

1.3 Autenticazione

Un'operazione di sistema secondaria, è l'integrazione dell'autenticazione attraverso diversi social network: Google, Linkedin, Amazon.

Per poter integrare l'autenticazione, è necessario interagire con le sorgenti esterne (Google, Linkedin, Amazon), in particolare per verificare le credenziali d'accesso dell'utente, ed estrarne le informazioni personali.

Nell'operazione di sistema di autenticazione, l'utente clicca sul bottone login, inserisce le credenziali, e le invia al sistema.

Il sistema allora verifica le credenziali, attraverso la sorgente esterna scelta dall'utente.

Se le credenziali sono corrette procede con un'ulteriore interazione con la sorgente esterna per acquisire le informazioni personali dell'utente e mostrandole, altrimenti mostra un messaggio di errore.

Quando un utente effettua l'autenticazione attraverso uno dei social disponibili per la prima volta, il sistema memorizza le informazioni riguardanti l'utente.

L’utilizzo del sistema viene descritto in modo dettagliato attraverso il seguente caso d’uso:

Caso d’uso UC3: Autenticazione tramite google/amazon/linkedin

1. L’utente clicca sulla scritta “login”.
2. Il sistema indirizza l’utente sulla form per autenticarsi tramite google/linkedin/amazon.
3. L’utente inserisce le sue credenziali.
4. Il sistema verifica le credenziali inserite e torna alla pagina principale mostrando in alto a destra l’immagine profilo associata all’utente ed il nome. L’icona resta presente in tutte le pagine di Glimpse finché la connessione è attiva.

4a. Il sistema risponde indicando che le credenziali sono errate nella finestra e non la chiude.

Essendo presente un’operazione di sistema per effettuare il login, ovviamente ne è presente un’altra per il logout.

L’operazione di sistema per il logout, risulta molto semplice.

Quando l’utente clicca sulla propria immagine del profilo, viene mostrato un menù a tendina dove è presente un bottone per effettuare il logout.

Quando l’utente clicca sul bottone, il sistema disconnette l’utente e rimuove le informazioni associate.

L’utilizzo del sistema viene descritto in modo dettagliato attraverso il seguente caso d’uso:

Caso d’uso UC4: Logout dal sistema

1. L’utente clicca sull’icona associata la suo account.
2. Il sistema mostra l’opzione “logout”.
3. L’utente clicca su “logout”.
4. Il sistema disconnette l’utente e fa scomparire l’icona associata.

Anche in questo caso come per gli importatori (descritti nel capitolo 1.1), essendo i concetti molto simili tra loro risulta conveniente effettuare una generalizzazione,

avendo una superclasse con le informazioni in comune, estesa dalle sottoclassi, una per ogni autenticazione.

In Figura 2 troviamo una parte del modello di dominio che rappresenta la generalizzazione:

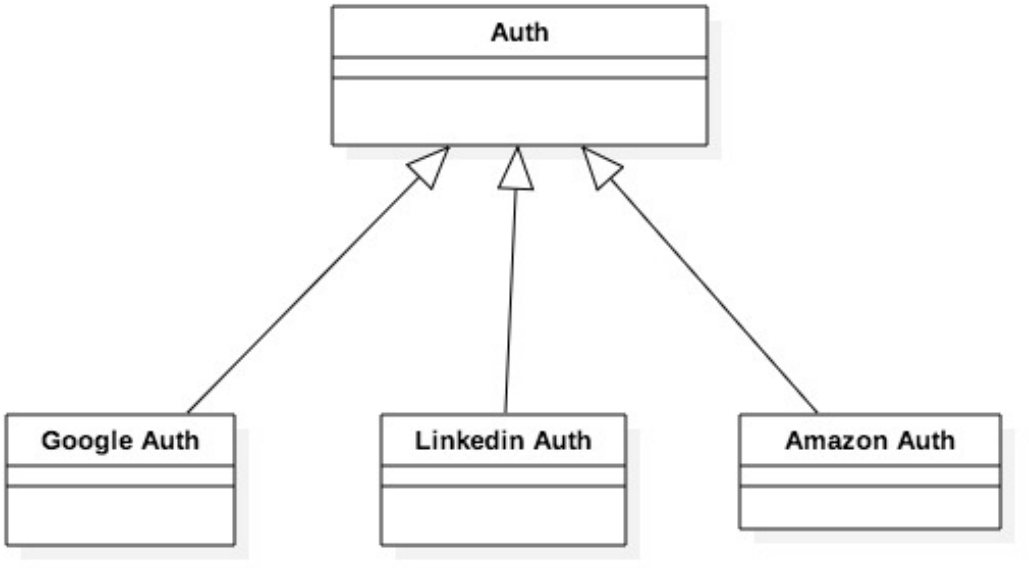

Figura 2. Parte del modello di dominio che rappresenta la generalizzazione degli autenticatori

1.4 Autorizzazione

Un'altra operazione di sistema secondaria è l'autorizzazione ad utilizzare le singole serie.

L'applicazione è destinata ad una monetizzazione futura, in particolare viene resa a pagamento l'esportazione delle serie, è quindi necessario effettuare un controllo sulle serie.

Glimpse suddivide le sorgenti in due categorie:

1. Sorgenti gratuite
2. Sorgenti a pagamento

In particolare un utente può esportare sempre una serie storica appartenente ad una sorgente gratuita, differentemente può esportare una time series appartenente ad una sorgente a pagamento solamente se è in possesso di una tipologia di abbonamento. Gli abbonamenti si suddividono in due categorie:

1. Abbonamenti temporali: la tipologia di abbonamenti che ha una data di scadenza fissata.
2. Abbonamenti pay-as-you-go: questa tipologia, mette a disposizione un certo numero di elementi da poter esportare. Per esempio un utente effettua un abbonamento pay-as-you-go al prezzo di 10 euro, quindi il contatore degli elementi del proprio abbonamento viene incrementato di 10 unità. Successivamente effettua l'esportazione di una serie a pagamento, quindi il suo abbonamento pay-as-you-go scende a 9 elementi.
L'abbonamento sarà scaduto quando il contatore degli elementi da poter scaricare sarà pari a 0.

Le informazioni sono descritte in modo dettagliato dal presente caso d'uso:

Caso d'uso UC5: Accesso a serie di sorgenti a pagamento

1. L'utente chiede di esportare una serie relativa ad una sorgente a pagamento.
2. Il sistema verifica che l'utente abbia a disposizione un abbonamento temporale non scaduto, oppure un pay-as-you-go non esaurito e, nel secondo caso, scala un'unità dalle serie disponibili, se viene eseguito l'esportazione.

2. Tecnologie e formati utilizzati

Il Capitolo 2 tratta della metodologia di lavoro utilizzata per lo sviluppo dell'applicazione, ovvero l'utilizzo del framework Scrum, e dei requisiti di sistema, ovvero le tecnologie utilizzate all'interno del progetto, tra le quali MongoDB, ElasticSearch, SQLite.

2.1 Framework Scrum

Nel progetto abbiamo utilizzato Scrum [12], è un framework agile di sviluppo del software, iterativo ed incrementale, concepito per gestire progetti e prodotti software o applicazioni di sviluppo.

Scrum enfatizza gli aspetti di gestione di progetto legati a contesti in cui è difficile pianificare in anticipo.

Esso suddivide il lavoro in degli *sprint*, la cui durata è fissa, e definita all'inizio del progetto, solitamente settimanale, bisettimanale o mensile, gli sprint sono fissi, è possibile terminare prima, ma non è possibile allungarlo, e giornalmente viene aggiornata la stima delle ore mancanti al termine di ogni *items*.

Un *item*, rappresenta una parte del prodotto finale da sviluppare, al quale viene associata una priorità, più è alta e più risulta importante, ed una stima di ore per svilupparlo.

All'inizio di ogni *sprint* ogni sviluppatore sceglie degli *items* disponibili, ove non è possibile modificarne il contenuto a metà dello sprint.

Ogni giorno viene effettuata una riunione con tutti i membri del team, che viene chiamata "*Daily Scrum*", e si discute ciò che è stato fatto il giorno prima, ciò che si dovrà fare, e se sono presenti eventuali blocchi.

È presente un "*Product Backlog*", che è una lista degli *items* disponibili, ordinate per priorità, in base alle prorità del cliente, ed evolve durante lo sviluppo del progetto. Ogni elemento appartenente al *backlog*, ha 5 campi che lo rappresentano:
1. Descrizione del problema

2. Priorità: che solitamente va da 1 a 7, dove 1 rappresenta la priorità massima e 7 la priorità minima.

3. Ore stimate per la risoluzione.

4. Un campo che definisce se è possibile prendere in carico l'*item*.

5. Un campo che definisce se l'*item* è stato già implementato.

Lo Scrum Team è suddiviso in 3 ruoli principali:

- Scrum Master: Il suo compito principale è aiutare il Team, con i vari problemi che possono essere sollevati, inoltre è anche di aiuto al Product Owner, per realizzare un progetto migliore.
  Esso non è visto come un leader, ma più come una guida.
- Team: Ogni persona è un membro del team, non ha un ruolo specifico, il suo obiettivo è raggiungere l'obbiettivo settimanale dello sprint, ha molta flessibilità su come sviluppare, e spesso può suggerire al Product Owner come migliorare il prodotto finale.
- Product Owner: Il suo compito è massimizzare il guadagno, identificando le priorità e modificando continuamente il Product Backlog, interagisce con l'utente finale per le decisioni.

Inoltre secondo il framework Scrum, viene definito un DOD ("*Definition of Done*"), ovvero una serie di passaggi, che vanno seguiti nella realizzazione dei singoli *item*, che può essere definito terminato quando tutte le istruzioni del DOD sono soddisfatte.

L'obiettivo dell'utilizzo di Scrum è garantire una trasparenza. Permette inoltre di evidenziare impedimenti sollevati sia per il team che per il Product Owner. Per esempio, se al primo sprint il team non riesce a rispettare le previsioni effettuate, non viene visto come un fallimento, ma più come un segnale di ridurre il carico al prossimo sprint.

2.2 ElasticSearch

ElasticSearch [9] è un server di ricerca con capacità Full Text, con supporto ad architetture distribuite. Tutte le funzionalità sono nativamente esposte attravero un'interfaccia RESTful, mentre le informazioni sono gestite come documenti JSON.

Può essere utilizzato localmente, con un server dedicato, o altrimenti l'azienda produttrice offre un servizio a pagamento che permette di utilizzare dei server dedicati. ElasticSearch è considerato come il secondo motore di ricerca più popolare, ed è utilizzato da grandi compagnie, quali per esempio eBay.

Esso può essere usato per cercare qualsiasi tipo di documento e fornisce un sistema di ricerca scalabile, è distribuito quindi gli indici possono essere suddivisi in shard, ognuno con possibilità di replica. Routing e bilanciamento sono effettuati automaticamente.

Mette a disposizione delle API REST che rendono molto semplice interrogarlo e creare documenti al suo interno.

In particolare permette di caricare un elemento al suo interno effettuando una chiamata http con metodo PUT, passando come body della chiamata un file json che rappresenta il documento da inserire. Stessa cosa vale per le altre operazioni, per interrogarlo viene effettuata una chiamata http con metodo POST, e la cancellazione con metodo DELETE.

Mette a disposizione anche delle "Bulk API", che permettono di caricare o cancellare più documenti con una singola chiamata http.

In particolare vengono inseriti tutti i documenti che si intende aggiungere all'indice in un file ".elastic", un documento è ad esempio il seguente:

```
{"create" :
       { "_index" : "timeseries",
       "_type" : "external",
       "_id" : "fred.all.00XALCATM086NEST" }
}
{"id" : "fred/all/00XALCATM086NEST",
"name" : "Harmonized Index of Consumer Prices: Overall Index Excluding Alcohol and Tobacco for Austria©",
"description" : "Harmonized Index of Consumer Prices: Overall Index Excluding Alcohol and Tobacco for Austria©",
"source" : "FRED", "country" : "ALL" }
```

e successivamente eseguire un'unica chiamata http che inserirà tutti i documenti nell'indice, un esempio di chiamata è la seguente:

```
curl -s -XPOST localhost:9200/_bulk --data-binary@glimpse_ts/indexer/FRED.elastic; echo
```

Questa tecnica risulta notevolmente superiore rispetto all'aggiungere un elemento alla volta, poiché la scrittura in un file è molto più veloce rispetto all'inserimento di un

documento in Elastic, e il caricamento Bulk, risulta istantaneo, anche con un numero molto grande di elementi

Inoltre ha moltissime funzionalità, per esempio ad ogni elemento restituito dalla ricerca è assegnato uno score automatico, che va da 0 a 1, e sarà tanto alto quanto è probabile che l'elemento trovato sia quello desiderato dall'utente.

2.3 MongoDB

MongoDB [11] è un DBMS non relazionale, esso si allontana dalla classica struttura dei database tradizionali basata su tabelle, in favore di documenti stile JSON (MongoDB chiama il formato BSON), rendendo l'integrazione di dati per alcune applicazioni più facile e veloce.

È utilizzato come backend da molti dei grandi siti web come eBay e New York Times. MongoDB supporta ricerche per campi, intervalli, e regular expression. Le query possono restituire campi specifichi del documento.

Qualunque campo può essere indicizzato, con indici unici, secondari, sparsi, geospaziali e indici full text.

Fornisce alta disponibilità e aumento del carico gestita attraverso i replica set. Un replica set consiste in più copie dei dati. Ognuna di esse può avere il ruolo di copia primaria o secondaria. Quella primaria effettua tutte le scritture e le letture. Quelle secondarie mantengono una copia dei dati della replica primaria.

Quando una replica primaria fallisce, il replica set innesca un processo di elezione per determinare quale delle repliche secondarie deve diventare una replica primaria. I dati all'interno sono suddivisi in "*Collection*", che a sua volta è composta da *Document*, che rappresentano degli elementi.

Scala orizzontalmente usando lo sharding. L'utente deve scegliere una chiave di sharding, che determina come i dati di una collection saranno distribuiti.

I dati sono divisi in intervalli e distribuiti su molteplici shard.

Include anche un meccanismo di bilanciamento dei dati, se per esempio uno shard risulta troppo carico ed un altro risulta vuoto, sposta i dati da quello carico a quello vuoto, in modo da bilanciare la distribuzione dei dati all'interno del cluster. MongoDB ha i driver ufficiali per i più popolari linguaggi di programmazione.

2.4 SQLite

SQLite è una libreria software che implementa un DBMS SQL. Non è un processo standalone utilizzabile di per sé, ma può essere incorporato all'interno di un altro programma.

Una delle caratteristiche principali di SQLite è che risulta molto veloce.

2.5 Formato SDMX-ML

Il formato SDMX [13] (*Statistical data and metadata eXchange*), è un formato ISO standard, ideato per descrivere dati statistici e metadati.

Il formato SDMX fornisce i seguenti vantaggi:

1. Facilitare lo scambio di dati e metadati
2. Fare un uso efficiente delle tecnologie e degli standard
3. Migliorare la disponibilità di dati statistici e di metadati per l'utente
4. Diffusione dei dati

Il formato SDMX viene utilizzato da diverse sorgenti: Eurostat, UN ("*United Nations*"), BIS, World Bank, OECD("*Organisation for Economy Co-operation and Development*"), ECB.

Il formato SDMX fornisce due standard per essere rappresentato: SDMX-JSON che utilizza la sintassi JSON e SDMX-ML che utilizza la sintassi XML.

Sono presenti diversi schemi per il formato, ogni schema rappresenta informazioni differenti.

Gli schemi più importanti sono il DSD ("*Data Structure Definition*") e il file che rappresenta le osservazioni.

Le time series nel formato SDMX vengono rappresentate attraverso dei dataset, contenenti più time series, che vengono estratte attraverso delle particolari query, utilizzate per identificarle.

Il DSD definisce come sono strutturati i dati di un dataset, al suo interno vengono descritti dei campi. In particolare i campi sono i seguenti:

1. *Key Families*: descrive come è diviso il dataset, contiene una lista di elementi chiamati "*Dimension*", che descrivono quali campi è necessario inserire nella query per estrarre una serie storica. Inoltre sono presenti anche altri elementi chiamati

"*Attribute*", anch'essi descrivono dei campi da inserire nella query, ma al contrario delle dimensioni, sono facoltativi.

2. *Concepts*: fornisce una descrizione, solitamente in diverse lingue, per tutte le *dimension* e tutti gli *attribute*.

3. *CodeLists*: rappresenta una lista che descrive i valori che è possibile associare ad ogni *Dimension* e ad ogni *Attribute.*

È possibile estrarre una *time series,* fornendo i valori di ogni dimensione presente nel DSD.

Le serie sono definite all'interno di un file, dove sono presenti diversi campi: è presente un campo "*SeriesKey*", dove vengono descritte le dimensioni ed i valori utilizzati, ed una lista di campi "*Obs*", che rappresentano appunto le singole osservazioni, quindi forniscono una data ed un valore.

2.6 Oauth2.0

OAuth2.0 [14] è un protocollo di comunicazione mediante il quale un'applicazione può gestire in modo sicuro l'accesso ai dati sensibili. È applicabile con qualsiasi tipo di applicazione: Desktop, Web, Mobile.

Viene adottato dalla maggior parte delle aziende che offrono servizi web: Google, Facebook, Linkedin, Amazon, ecc...

La procedura definita dal protocollo OAuth2.0 è la seguente:

1. Per poter usufruire del servizio è necessario iscrivere l'applicazione ad i vari social di interesse, ricevendo delle credenziali chiamate "*client_ID*" e "*client_secret*", senza le quali non è possibile usufruire del servizio.

2. Quando l'utente si autentica viene restituite un codice, chiamato "*authorization code*", con tempi di scadenza molto brevi

3. Il sistema deve immediatamente effettuare l'operazione di scambio del codice di autorizzazione, ovvero attraverso una chiamata http, inserendo come *query parameter,* le credenziali dell'applicazione ed il codice, ritorna il "*token*", quest'ultimo ha tempi di scadenza più lunghi rispetto al codice di autorizzazione.

4. Una volta ottenuto il token è possibile utilizzarlo per ottenere informazioni dell'utente, come l'immagine del profilo, l'email, il nome, il cognome, ecc...

5. Dopo che il token è scaduto, è possibile aggiornarlo e ricevere un nuovo token, questa operazione può essere eseguita un massimo di 25 volte.

3. Estrazione e persistenza dei dati da una fonte

Il Capitolo 3 tratta la progettazione dell'estrazione e la persistenza dei dati attraverso la sorgente World Bank.

3.1 Estrazione dati da World Bank

L'estrazione dei dati da World Bank [1] avviene utilizzando delle API REST (*Representional State Transfer*), che si basano sull'utilizzo dei metodi del protocollo http per effettuare tramite semplici richieste al server le operazioni CRUD (*Create*, *Read*, *Update*, *Delete*).

Un esempio di chiamata è la seguente:

http://api.worldbank.org/countries/paese/indicators/nome_timeseries?format=JSON

Dove per paese si passa l'iso code, per esempio "ITA", e per il nome_timeseries il nome della *time series*.

World Bank restituisce i dati in formato json, in Python risulta molto semplice effettuarne il parse, in quanto una funzione base denominata eval, esegue la trasformazione da json ad un dizionario

L'operazione si divide in due casi:

- Ricerca della serie storica: dove l'utente effettua la ricerca e se la serie esiste vengono mostrati i metadati, come rappresentato in Figura 3.

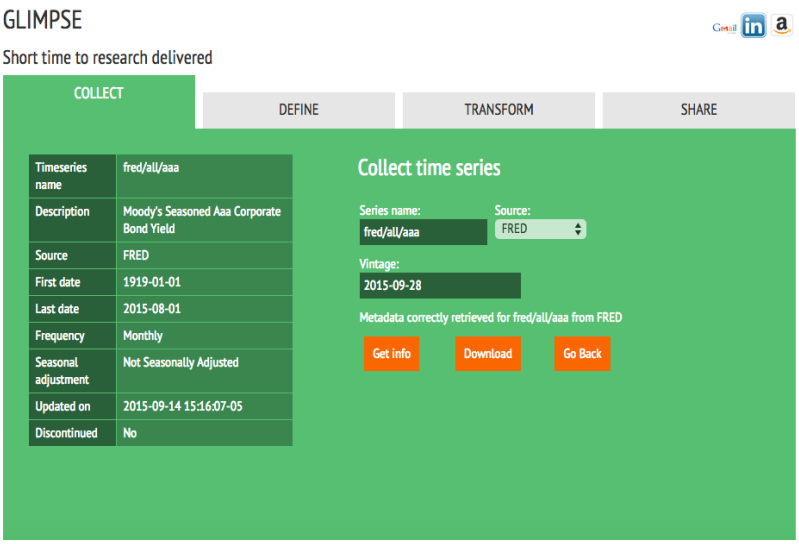

Figura 3. Schermata dell'applicazione che rappresenta i metadati

- Download della serie storica: dopo aver trovato la serie è possibile effettuarne il download.

È stata creata una classe "WorldBankImporter" per la gestione dell'estrazione dei dati. In tale classe sono presenti diversi metodi per eseguire sia l'operazione di ricerca di una *time series*, che l'operazione di download.

Innanzitutto è presente un metodo utilizzato per effettuare la chiamata http alle API, tale metodo utilizza il modulo urllib2 di Python, che offre la funzionalità di aprire la *response* restituita dalla chiamata. Vengono inoltre gestite le eccezioni, se la chiamata dovesse sollevare gli errori del protocollo http.

```
def do_rest_call(self, url):
    try:
        log.info("HTTP Request: " + url)
        response = urllib2.urlopen(url).read()
        return response
    except urllib2.HTTPError as ex:
        if ex.code == 404 or ex.code == 400:
            raise ImportException("Timeseries not found.")
        if ex.code == 500:
            raise ImportException("Vintage not available.")
        log.error("HTTP Error: " + str(ex.code) + "  " +
str(ex.reason))
        raise ImportException(str(ex.reason))
```

Le API di World Bank forniscono dei metadati in modo diverso rispetto a come sono definiti all'interno di Glimpse. Quindi sono stati definiti dei metodi di supporto per cercare e trasformare i metadati nel formato di Glimpse.

Se la serie storica è annuale World Bank fornisce le date delle osservazioni con la stringa "2015", se quadrimestrale "2014Q01", ovvero il primo quadrimestre dell'anno 2014, se mensile "2015M01", ovvero il primo mese dell'anno 2015, mentre nel formato di Glimpse le date delle osservazioni sono definite con una stringa del tipo "2015-01-01".

Quindi è stato progettato un metodo di supporto, che permette la trasformazione della data dal formato fornito da World Bank al formato utilizzato in Glimpse, definendo l'anno ed il mese corrispondente, e poiché World Bank non fornisce serie storiche giornaliere, supponiamo il primo giorno del mese.

```
def __get_clean_date(self, date):
    if 'M' in date:
        return date.replace("M","-")
    elif 'Q' in date:
        if date[5:] == "01":
            return date[:4]+"-03"
        elif date[5:] == "02":
            return date[:4]+"-06"
        elif date[5:] == "03":
            return date[:4]+"-09"
```

```
            else:
                return date[:4]+"-12"
        else:
            return date+"-01"
```

Viene sollevato anche il problema della frequenza, restituendo le date delle osservazioni come definito, è necessario progettare un altro metodo di supporto che permette di determinare la frequenza della *time series*.

Il metodo risulta molto semplice in quanto occorre verificare nelle date delle osservazioni quale lettera è presente.

```
def __get_frequency(self, date):
    if 'M' in date:
        return "Monthly"
    elif 'Q' in date:
        return "Quarterly"
    else:
        return "Annual"
```

La chiamata http effettuata per estrarre la serie non fornisce la descrizione della serie storica, ma le API forniscono una chiamata differente che fornisce la descrizione.

È stato quindi progettato un altro metodo di supporto che permette di estrarre la descrizione della *time series,* risulta molto semplice, esegue la chiamata http che ritorna i dati in formato JSON, esegue il parse in un dizionario attraverso una funzione di base Python, chiamata eval, e ritorna il campo che contiene la descrizione della serie storica.

```
def get_title(self, tsName):
    url =
"http://api.worldbank.org/indicators/"+tsName+"?format=json"
    response=self.do_rest_call(url)
    return eval(response)[1][0]["sourceNote"]
```

È presente una classe *time series* che ha 2 variabili d'istanza, i metadati che sono rappresentati da un dizionario, e i dati che sono rappresentati da una lista di dizionari.

È stato quindi definito il metodo per ricercare una *time series,* che utilizzando i metodi di supporto definiti, salva tutti i metadati all'interno dell'oggetto *time series,* che verranno poi mostrati all'utente.

In Figura 4 viene mostrato il diagramma di interazione riguardante il metodo:

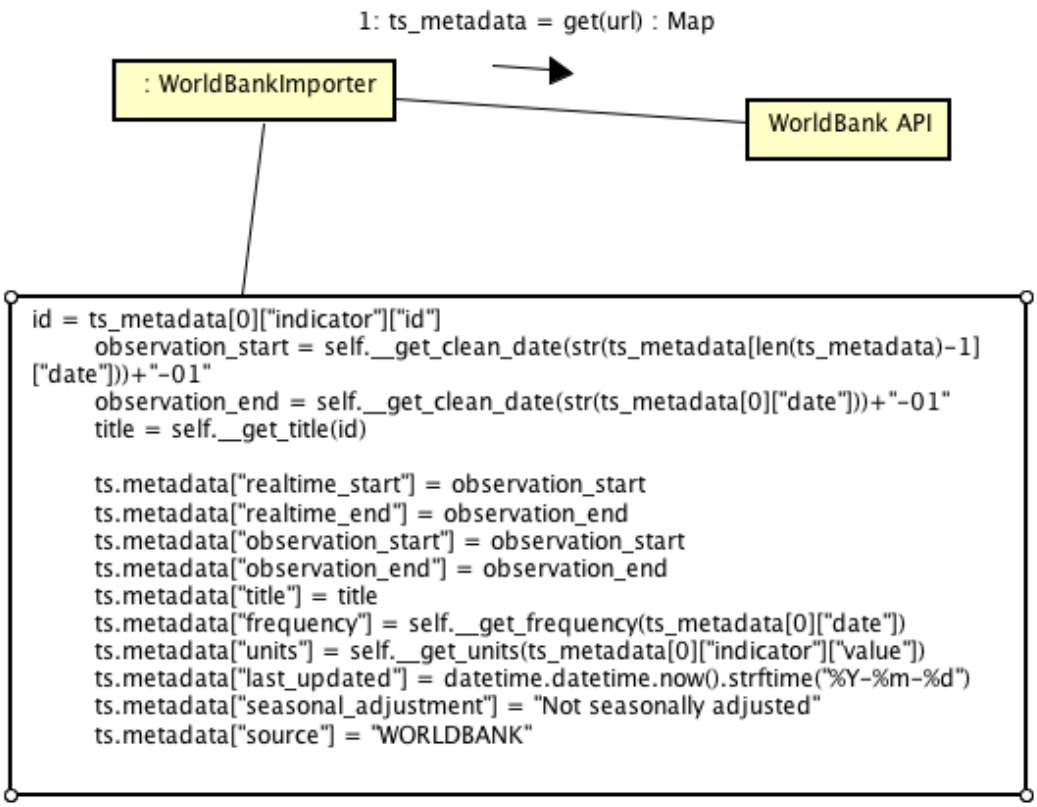

Figura 4. Diagramma di interazione del metodo set_ts_metadata di WorldBankImporter

Successivamente dopo aver effettuato la ricerca è possibile effettuare il download della *time series*.

Per effettuare questa operazione viene innanzitutto invocato il metodo appena definito per raccogliere i metadati e viene definito un altro metodo per effettuare l'estrazione delle osservazioni della serie storica.

È presente una classe Vintage, che offre la funzionalità di verifica se il vintage è nel formato corretto.

Nella progettazione del metodo viene inizialmente effettuata la chiamata http, viene effettuato un controllo sul vintage ed infine viene effettuata un'iterazione sulle osservazioni presenti, e vengono salvate una per una all'interno dell'oggetto *time series*.

In Figura 5 viene mostrato il diagramma di interazione riguardante il metodo:

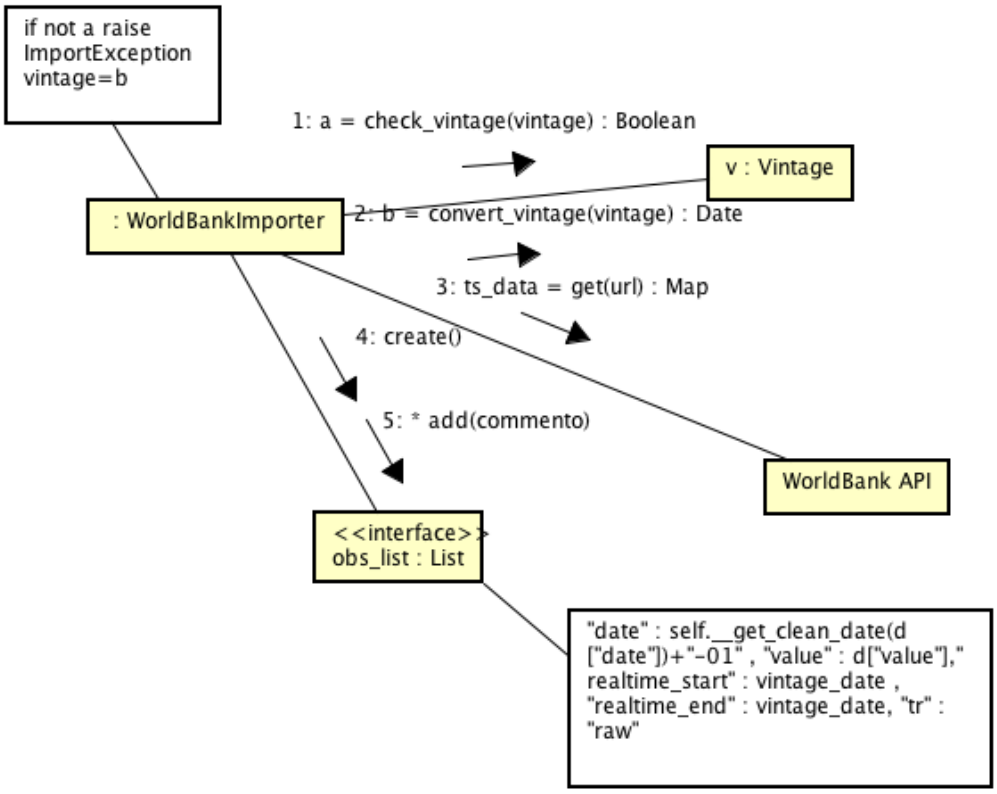

Figura 5. Diagramma di interazione del metodo set_ts_data della classe WorldBankImporter

Glimpse offre molte sorgenti dal quale estrarre dati, è stato quindi conveniente creare una gerarchia di classi, avendo una superclasse astratta denominata "Importer", che sarà estesa da tutte le classi che rappresentano le sorgenti.

In particolare, la classe "Importer", avrà il metodo per effettuare la chiamata http, poiché risulta in comune per tutte le sorgenti e i metodi per la ricerca e per il download, dove vengono effettuate le operazioni in comune fra tutte le sorgenti nella superclasse e vengono invocati altri metodi astratti implementati nelle sottoclassi.

3.2 Persistenza dei dati

L'oggetto *time series,* oltre ad avere le due variabili d'istanza, contiene anche molti metodi per la gestione delle *time series.*

Uno dei metodi più importanti è il metodo "persist", che si occupa della persistenza delle *time series* all'interno di MongoDB.

Quando un utente esegue il download di una *time series* è necessario memorizzarla all'interno del database, quindi dopo aver eseguito il metodo che effettua il download, viene eseguito il metodo persist. Questo metodo utilizza principalmente il modulo

pymongo utilizzato per la gestione di MongoDB da Python, che offre funzionalità di base, ad esempio le operazioni CRUD.

Il metodo persist verifica se la *time series* non sia già presente all'interno di MongoDB, nel caso fosse presente esegue un merge, ovvero unisce le osservazioni dell'occorrenza già presente con le osservazioni di quella nuova, mantenendole entrambe, nel caso in cui non fosse presente mantiene la nuova occorrenza richiesta.

In ogni caso infine viene eseguita l'operazione di inserimento in MongoDB.

All'interno di MongoDB è presente una *Collection* denominata *Timeseries*, che è composta da *Document,* strutturati al seguente modo:

- Object Id: chiave univoca che viene generata da MongoDB

- Metadata: rappresenta tutti i metadati riguardanti la *time series*. Un esempio è il seguente:

```
"metadata": {
        "last_updated" : "2015-08-27 08:51:09-05",
        "observation_start : "1946-01-01",
        "title" : "Gross Domestic Product",
        "seasonal_adjustment" : "Seasonally Adjusted Annual Rate",
        "observation_end" : "2015-04-01",
        "realtime_end" : "2015-09-23",
        "id" : "fred/all/gdp",
               "source" : "FRED",
         "frequency" : "Quarterly",
               "units" : "Billions of Dollars",
         "source_name" : "Federal Reserve Bank of St. Louis",
               "realtime_start" : "2015-09-18",
               "discontinued" : "No"
}
```

- Data: contiene una lista con tutte le osservazioni. Un esempio breve contenente soltanto due osservazioni è il seguente:

```
"data": [
        {
            "date" : "1946-01-01",
            "tr" : "raw",
            "realtime_start" : "2015-09-23",
            "realtime_end" : "2015-09-23",
            "value" : "."
        },
        {
```

```
    "date" : "1946-04-01",
    "tr" : "raw",
    "realtime_start" : "2015-09-23",
    "realtime_end" : "2015-09-23",
    "value" : "."
}]
```

4. Estrazione dei dati da altre fonti

Il Capitolo 4 tratta dell'estrazione e la persistenza dei dati dalle altre sorgenti, il modo di procedere è simile a quello utilizzato per World Bank, ma con il cambiamento di alcuni dettagli.

4.1 Estrazione dei dati da Eurostat

Eurostat [2] come World Bank, mette a disposizione delle API REST, che permettono di estrarre i dati.

A differenza di World Bank, fornisce i dati nel formato SDMX-ML (descritto nel capitolo 2.5).

Un esempio di chiamata è la seguente:

http://ec.europa.eu/eurostat/SDMX/diss-web/rest/data/nome_dataset/dim

Dove nome_dataset sta per il nome del dataset e dim sta per le dimensioni specificate per identificare la *time series.*

Diversamente da come implementato per World Bank, utilizzando il modulo urllib2, viene sollevato un errore interno del server, un Errore http 500.

È stato risolto questo problema utilizzando il modulo pycurl, che permette di aggirare il problema.

```
def do_rest_call(self, url):
    try:
        log.info("HTTP Request: " + url)
        results = BytesIO()
        c = pycurl.Curl()
        c.setopt(c.URL, url)
        c.setopt(c.WRITEDATA, results)
        c.perform()
        c.close()
        return results.getvalue()
    except:
        raise ImportException("Timeseries not found.")
```

Anche il parse dei dati è stato effettuato in modo differente, essendo un file SDMX-ML, è stata utilizzata la libreria xml.etree.ElementTree, che permette di analizzare il file in maniera semplice

4.2 Estrazione dei dati da DOE (Department of Energy)

I dati DOE vengono estratti da una piattaforma denominata "Quandl" [5], anch'essa mette a disposizione delle API REST, e restituisce le informazioni in formato json.

Fornisce i dati in forma migliore rispetto a World Bank, poiché è possibile fare chiamate distinte, a seconda se si è interessati ai dati o ai metadati, quest'ultimi sono praticamente identici al formato utilizzato in Glimpse, quindi risulta molto semplice.

Quandl permette di estrarre i dati gratuitamente, ma con un certo limite, occorre una chiave particolare denominata APIKEY, per utilizzare il servizio senza limiti, ottenerla è molto semplice, basta registrarsi alla piattaforma.

Un esempio di chiamata per estrarre i metadati da Quandl è:

https://www.quandl.com/api/v3/datasets/DOE/nome_timeseries/metadata.json?api_key=APIKEY

Dove il "nome_timeseries", coincide con il nome della *time series* di interesse.

Un esempio di chiamata per estrarre i dati da Quandl è:

https://www.quandl.com/api/v3/datasets/DOE/nome_timeseries/data.json?order=asc&api_key=APIKEY

Che ordina le osservazione dalla prima all'ultima con il query parameter order=asc.

4.3 Estrazione dei dati da OPEC

Opec [4], che sta per "*Organization of the Petroleum Exporting Countries*", è una sorgente un po' diversa rispetto a tutte le altre, mentre le altre offrono un numero molto elevato di *time series*, essa ne offre soltanto una, in particolare rappresenta il prezzo al barile del petrolio al variare del tempo.

Tale *time series* viene fornita in formato XML, e quindi ho utilizzato il modulo xml.etree.ElementTree, per effettuare il parse del documento.

Un esempio di chiamata per estrarre la *time series* da Opec è:
http://www.opec.org/basket/basketDayArchives.xml

Si semplifica notevolmente, la raccolta dei metadati, poiché sono sempre fissi, e risulta facile gestirli, mentre per i dati viene effettuata sempre la chiamata, poiché sono aggiornati giornalmente.

4.4 Estrazione dei dati da BIS

BIS sta per "*Bank for International Settlements*", è una sorgente molto particolare, poiché non mette a disposizione nessuna interfaccia REST, ma restituisce tutte le *time series* disposte in colonna in un file Excel.

All'interno di questo file Excel, sono presenti 3 fogli, dove vengono suddivise le serie annuali, quadrimestrali e mensili.

All'interno di questo file le serie non sono identificate da un id come tutte le altre *time series,* ma da una descrizione.

È stato quindi deciso di prendere come identificativo della serie storica la pagina e la colonna excel associata alla particolare serie, per esempio la serie "m.c", riferisce alla terza colonna della pagina delle serie mensili.

Per risolvere il problema di come estrarre i dati, abbiamo cambiato totalmente approccio, poiché questo file excel risulta pesante e rallenterebbe le prestazioni, abbiamo deciso di non effettuare la chiamata http ogni volta che l'utente cerca una serie, ma di salvare in locale il file ed aggiornarlo in background, un numero sufficiente di volte.

Con questo approccio, basta effettuare un accesso al file excel ed analizzarlo con un parser.

Ho utilizzato una libreria Python "xlrd", che permette di effettuare il parse di un file excel, e fornisce funzionalità per analizzarlo in modo semplice e veloce.

5. Gestione della ricerca: ElasticSearch

Il Capitolo 5 descrive il motore di ricerca full-text sviluppato per serie storiche per la data dashboard dell'applicazione.

5.1 Scelta dell'indice

La scelta iniziale per l'indice da utilizzare è stata quella di usare Whoosh.

Whoosh [10] è una libreria scritta in Python, che permette la creazione di indici. Abbiamo eseguito test su questo particolare indice, ed abbiamo riscontrato molte problematiche:

- Non supporta una grande mole di dati.
- Molto lento nell'inserimento di documenti all'interno.
- Non offre uno score che rappresenta la probabilità che il documento sia quello ricercato.
- Genera un file, che deve essere necessariamente all'interno del progetto.
- Non è possibile fare query che mostrano tutto l'indice
- Risulta complicata la costruzione della query, e poco efficiente

Quindi abbiamo deciso di abbandonare l'idea di utilizzare Whoosh come indice.

Successivamente abbiamo così deciso di utilizzare come indice ElasticSearch [9], che ha soddisfatto i nostri requisiti.

ElasticSearch è un prodotto open source che permette di creare un indice al suo interno, ed è appoggiato localmente in un server dedicato, la sua potenzialità è la capacità di effettuare query in tempi ottimali, e la velocità in quanto permette 2,8 richieste al secondo in un calcolatore molto potente.

In Figura 6 [15] viene rappresentato un benchmark effettuato su ElasticSearch, Whoosh e Solr, un'altra piattaforma utilizzata per la ricerca.

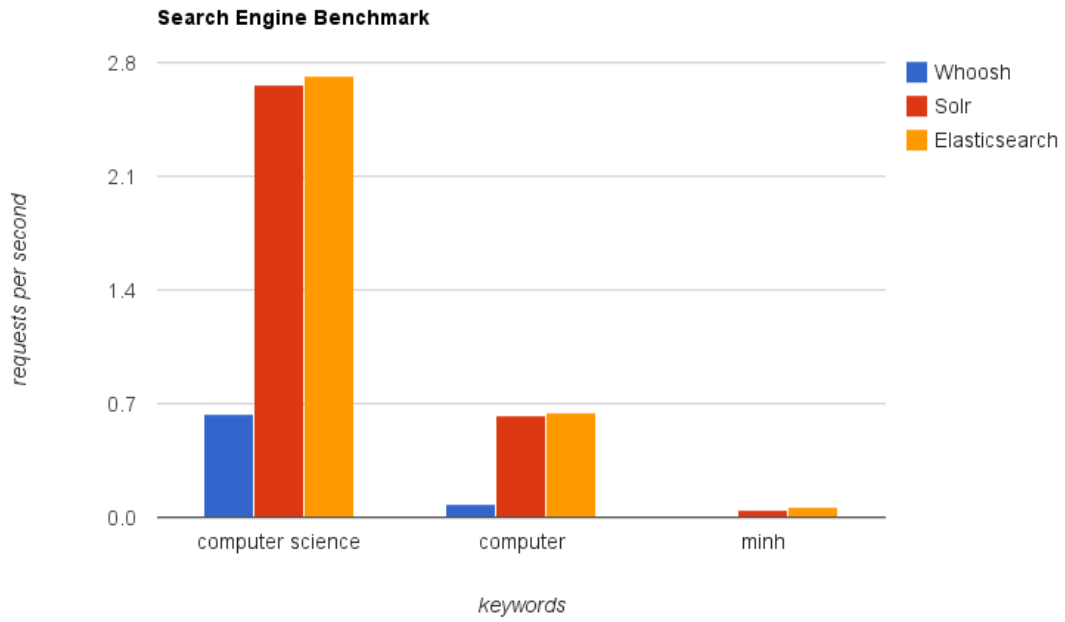

Figura 6. Benchmark effettuato su Elasticsearch, Whoosh, ed Solr

Con l'utilizzo di ElasticSearch, c'è stato un netto miglioramento della velocità, in particolare per l'accesso all'indice attraverso chiamate http, e la possibilità di utilizzare le BULK API, che permettono di inserire molti documenti insieme.

5.2 Struttura dell'indice

È necessario definire una struttura dell'indice, i documenti in ElasticSearch sono gestiti come file JSON, è pero possibile definire qualsiasi campo a piacere.

È stato deciso in Glimpse di utilizzare un unico indice chiamato "*timeseries*", che si suddivide in due parti, "*external*" ed "*internal*", rispettivamente per rappresentare le serie storiche che non sono state ancora scaricate, e quelle presenti in MongoDB.

Inoltre ogni serie nell'indice deve essere identificata da una chiave univoca, quindi è stata utilizzata la forma "sorgente.paese.nome_serie", non potendo usare il carattere '/' in una chiamata http.

Le serie storiche inserite nell'indice, sono composte da 6 campi, che rappresentano rispettivamente l'id della serie, il nome per esteso, la descrizione, i topic (gli argomenti di cui tratta la serie), la sorgente ed il paese, che nel caso non sia presente avrà il valore ALL.

Un esempio di documento di una serie storica nell'indice ElasticSearch è il seguente:

```
{
```

```
_index: "timeseries",
_type: "external",
_id: "worldbank.DO.2.0.cov.Math.pl_2.pub",
_score: 1,
_source:

{

  description: "The coverage rate is the childhood access rate of a given
  opportunity used in calculating the Human Opportunities Index (HOI). The
  coverage rate does not take into account inequality of access between
  different circumstance groups.",
  country: "DO",
  topic: "Poverty , ",
  source: "WORLDBANK",
  id: "worldbank/DO/2.0.cov.Math.pl_2.pub",
  name: "Coverage: Mathematics Proficiency Level 2, Public schools"

}

}
```


In questo caso viene rappresentata una serie storica appartenente a World Bank, è presente anche un campo "*score*", che determina la probabilità della serie ricercata, in questo caso risulta 1, poiché è stata effettuata una ricerca di tutte le serie storiche.

5.3 Creazione indice World Bank

Per l'implementazione della creazione dell'indice per World Bank, sono stati utilizzati diversi approcci.

Le *time series*, fornite da World Bank vengono identificate dal nome e dal paese richiesto.

Le API REST, non forniscono una chiamata che permette di prendere direttamente tutti i paesi esistenti per la singola serie storica.

È disponibile invece una chiamata http che permette di accedere a tutte le *time series* che fornisce World Bank, che sono circa 13000.

È inoltre possibile effettuare una chiamata http, che data una serie storica, fornisce in un unico file tutte le osservazioni per tutti i paesi, per effettuare questa chiamata è necessario inserire il parametro "ALL" nella parte della URL dove va inserito il paese.

Un esempio di chiamata è:

http://api.worldbank.org/countries/all/indicators/nome_serie

Avendo queste informazioni a disposizione, è stato progettato un primo approccio: attraverso il metodo della classe "WorldBankImporter", che permette di effettuare le

chiamate http, viene effettuata l'estrazione di tutte le serie storiche appartenenti a World Bank.

Per ognuna di queste viene effettuata un'ulteriore richiesta http, che fornisce un file con tutte le osservazioni di tutti i paesi per la serie.

Ogni file restituito viene analizzato per ottenere una lista dei paesi per la quale esiste la serie storica che combinata con i paesi trovati viene aggiunta all'indice.

Questo approccio risulta poco efficiente in quanto devo effettuare per ognuna delle 13000 serie una richiesta http, ed analizzarne una per una. Infatti i tempi di attesa risultano molto alti, circa 6 ore.

È stato quindi progettato un secondo e ultimo approccio, molto simile al primo approccio.

La differenza sta nel fatto che viene sfruttata la concorrenza per diminuire i tempi di attesa.

Attraverso la concorrenza è possibile effettuare più chiamate http in parallelo, ovvero invece di inviare una richiesta per volta ne vengono inviate un numero maggiore.

In questo approccio viene definito un numero di concorrenza, che equivale alle chiamate che vengono effettuate in parallelo, settato a 50.

Viene utilizzato un metodo che dopo aver eseguito la chiamata che restituisce tutte le serie storiche, costruisce delle stringhe contenenti le URL delle chiamate http da effettuare per ogni singola serie, e le memorizza in una lista.

Successivamente, vengono costruiti dei Thread pari al numero di concorrenza associato, i Thread rappresentano dei sottoprocessi che verranno eseguiti in parallelo.

Viene istanziata una coda (*Queue*), che viene popolata da tutti gli elementi presenti nella lista contenente le URL.

```
global q
print("Starting indexing WB")
q = Queue(concurrent * 2)
for i in range(concurrent):
    t = Thread(target=doWork)
    t.daemon = True
    t.start()
try:
    print("Getting all URLs")
    urls = get_all_urls()

    print("Creating index for WB")
    print("# of sources: " + str(len(urls)))

    for url in urls:
        q.put(url.strip())
    q.join()
```

```
        except KeyboardInterrupt:
            sys.exit(1)
```

Gli elementi della coda vengono lanciati in parallelo eseguendo un ulteriore metodo, che utilizza il modulo Python httplib, che permette di gestire le richieste per il protocollo http, ed il modulo pycurl.

In questo metodo viene effettuata la richiesta http per ogni elemento della coda, e viene analizzata la response per estrarre i paesi associati all'id della serie, ed infine vengono inserite nell'indice la serie storica combinata con tutti i paesi trovati, ne segue il codice riguardante questa parte del metodo.

```
c = pycurl.Curl()
for element in info:
    put_element = {"id": element,
                   "name": name,
                   "description": field_description,
                   "topic": field_topic,
                   "source": "WORLDBANK",
                   "country": element.split("/")[1]
                    }
    elastic_url =
"localhost:9200/timeseries/external/"+element.replace("/",
"."))+"?pretty"
    c.setopt(pycurl.URL, elastic_url)
    c.setopt(pycurl.POSTFIELDS, json.dumps(put_element))
    c.perform()
```

Utilizzando questo approccio, c'è stato un netto miglioramento dei tempi, in quanto è stato testato che l'algoritmo finisce in circa 30 minuti.

5.4 Creazione indice Eurostat

Eurostat fornisce circa 1 miliardo di serie storiche, quindi il problema principale per la creazione dell'indice risulta l'interfacciarsi con questa grande mole di dati.

Eurostat restituisce i dati in formato SDMX-ML (Descritti nel capitolo 2.5), fornendo dataset composti anche da 10 dimensioni.

È presente un servizio messo a disposizione da Eurostat [3], chiamato "Bulk Download", dove vengono forniti tutti i dataset in file zip.

In questo file è presente il DSD, ed un file contenente tutte le combinazioni delle dimensioni che forniscono serie storiche esistenti.

È stato quindi definito un metodo che estrae ogni dataset, e per ognuno, utilizzando anche le Bulk API di Elastic [9] (Descritte nel Capitolo 2.2), analizza il file SDMX

dove sono presenti le dimensioni del dataset, ed inserisce ogni combinazione nel file "EUROSTAT.elastic".

Le informazioni riguandanti la serie storica non sono presenti all'interno del file, quindi è stato definito un metodo di supporto, che restituisce la descrizione della serie analizzando il DSD.

In particolare utilizza il modulo Python urllib, che a differenza degli altri permette il download di file compressi, esegue la decompressione utilizzando il modulo zipfile, che permette di decomprimere i file SDMX che verranno poi analizzati attraverso il modulo xml.etree.ElementTree, per trovare le dimensioni.

In Figura 7 viene rappresentato il metodo che ritorna tutte le combinazioni delle dimensioni relative ad un dataset:

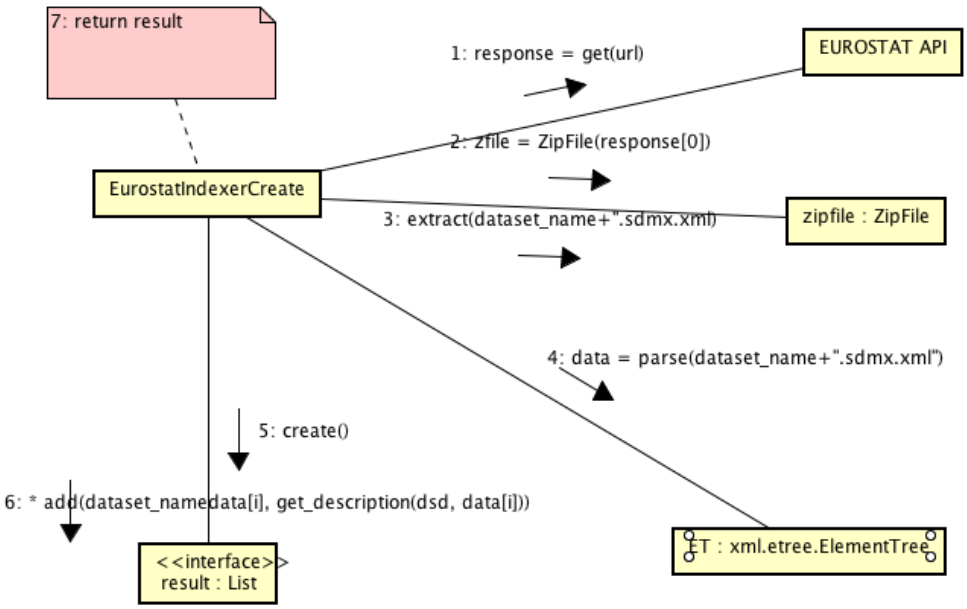

Figura 7. Diagramma di interazione riguardante il metodo che torna tutte le combinazioni delle dimensioni relative al dataset

Elastic permette di effettuare i caricamenti Bulk con un massimo di 100Mb, quindi è stato aggiunto al metodo un controllo, che verifica quando la dimensione del file elastic è maggiore di 100Mb, allora attraverso il modulo Python subprocess, che permette di eseguire operazioni in background, esegue la chiamata http per effettuare il caricamento, cancella il corrente file elastic e ne crea uno nuovo.

Eurostat fornisce circa 1 miliardo di serie storiche che equivalgono a 500Gb di dati.

Questo approccio è risultato molto efficiente, in quanto mantiene l'indice aggiornato ogni 100Mb e termina in circa 6 ore.

5.5 Creazione altri indici

Sono state definiti altri metodi per la creazione di due indici: BIS ed OPEC.

La creazione dell'indice per OPEC, è molto semplice, perché essendo una sorgente che gestisce soltanto una serie, è stato necessario sviluppare un metodo che aggiunge quell'unica serie all'indice Elastic.

La creazione dell'indice per BIS, è stata un po' più complicata.

Avendo tutti i dati all'interno di un file excel, ho eseguito la lettura di tutto il file attraverso il modulo Python xlrd, analizzando le pagine disponibili, ovvero delle serie mensili, quadrimestrali e annuali, sono state inserite una per una all'interno dell'indice, considerando sempre il formato "frequenza.colonna".

Essendo anche per BIS, poche serie, l'algoritmo risulta veloce, e funzionale.

5.6 Aggiornamento dell'indice

È possibile che le sorgenti aggiornino il proprio database, quindi un'operazione fondamentale è aggiornare l'indice periodicamente.

È stato sviluppato uno script, che permette di avviare i moduli per creare gli indici, che saranno eseguiti periodicamente in background, e se è presente qualche cambiamento l'indice Elastic lo aggiorna in automatico.

È possibile utilizzare questa procedura perché ElasticSearch, fornisce il riconoscimento delle versioni e dei cambiamenti, quindi se riceve lo stesso documento lo ignora, mentre se il documento ha un cambiamento di un parametro esegue la sostituzione, altrimenti se non è presente all'interno dell'indice, crea un nuovo documento.

Un caso particolare è la sorgente BIS, dato che i dati vengono presi in locale dal foglio

excel, quindi ogni qualvolta viene aggiornato l'indice, prima di effettuare l'operazione viene aggiornato il file excel, sostituendo la nuova versione con la precedente.

5.7 Interrogazione dell'indice

Lo scopo principale dell'integrazione di un indice è permettere la ricerca all'interno dell'applicazione, risulta quindi un'operazione fondamentale quella di interrogazione dell'indice.

Per l'interrogazione dell'indice sono importanti due campi dei documenti di ElasticSearch, nome e descrizione, che per alcune sorgenti possono essere uguali e per altre possono variare.

In ElasticSearch è possibile effettuare la ricerca attraverso una chiamata http con metodo POST, passando come corpo della chiamata un file JSON, dal quale viene stabilito il tipo di query che verrà effettuata.
La chiamata effettuata è la seguente:

```
localhost:9200/timeseries/external/_search?pretty
```

Ed un esempio di query è il seguente:

```
{
  "query": {
        "bool" : {
                "must" : [
                        {"match" : {"source" : "WORLDBANK"}}
                        ],
                "should" : [
                        {"match" : {"name" : "poverty"}},
                        {"match" : {"description" : "poverty"}}
                        ]
                }
        },
  "sort" : ["_score"],
  "size": 10
}
```

In particolare questo esempio ritorna i 10 elementi con score più alto, ordinati dal più alto al più basso, che provengono dalla sorgente "World Bank" e che hanno in nome o descrizione o in entrambi la parola "*poverty*".
Tutti gli elementi che si aggiungono nella lista "must", contengono necessariamente quei valori nei campi indicati all'interno dei documenti, mentre gli elementi aggiunti

nella lista “should”, non devono avere necessariamente la parola all’interno del nome o della descrizione, ad esempio la chiamata sopra può tornare una serie appartenente a World Bank, che ha “pover”, all’interno della propria descrizione.

Nel caso in cui la stringa completa non è presente nella descrizione, ma è presente una sottostringa, lo score risulta più basso, altrettanto se trova la parola “*poverty*” soltanto in nome e non in descrizione.

Per permettere la ricerca è stata creata una classe “IndexerSearch”, dove all’interno è stato definito un metodo che permette di effettuare la ricerca all’interno dell’indice in modo univoco per tutte le sorgenti.

Da una ricerca è possibile che venga un numero molto grande di risultati, quindi è stata inserita la paginazione.

Viene implementata attraverso la tecnologia “from size”, che viene offerta da Elastic.

È necessario inserire il parametro “from” nella query, ad esempio se “from” vale 0 e “size” vale 10, la chiamata tornerà gli elementi da 0 a 9 della lista, se “from” vale 10 e “size” vale 10, tornerà gli elementi da 10 a 19, e così via.

6. Autenticazione e Autorizzazione

Il Capitolo 6 tratta dell'autenticazione attraverso i social integrati all'interno dell'applicazione, e dell'autorizzazione.

6.1 Autenticazione

È necessario inserire la funzionalità di autenticazione in Glimpse attraverso i social: Google [6], Linkedin [7] ed Amazon [8], per permettere di gestire gli utenti e per l'autorizzazione all'esportazione delle *time series*.

Ho utilizzato per tutte le autenticazioni il protocollo Oauth 2.0 (descritto nel Capitolo 2.6).

Essendo tre tecnologie per l'autenticazione, è stata creata una superclasse astratta "Auth", che permette di gestire in modo migliore le sottoclassi.

Quindi sono state create tre sottoclassi che estendono la superclasse "Auth", per la gestione delle varie autenticazioni: "AmazonAuthentication", "GoogleAuthentication", e "LinkedinAuthentication", dove ognuna interagisce con le API dei singoli social.

Il token viene restituito in modo diverso rispetto al social utilizzato, ad esempio Google ed Amazon dopo che l'autenticazione è stata eseguita con successo tornano direttamente il token, viceversa Linkedin torna il codice di autorizzazione, quindi necessita di un passaggio in più.

Una volta ottenuto il token è possibile effettuare delle operazioni per estrarre le informazioni di interesse, in particolare sono di interesse l'immagine del profilo, il nome ed il cognome.

Ogni sottoclasse ha una tecnologia diversa per ottenere informazioni, quindi è stato definito un metodo astratto nella classe "Auth", che sarà implementato dalle sottoclassi.

Nel metodo implementato per ogni sottoclasse si effettua una chiamata http passando

nella query string il token, e nella response vengono restituite le informazioni richieste dell'utente.

Un esempio di chiamata è la seguente:

https://api.amazon.com/user/profile?access_token=token

Ed infine ritorna una tupla, ovvero l'unione di uno o più elementi, che viene rappresentata nell'ordine: nome e cognome, l'immagine del profilo e l'email.

Quindi si mantiene la connessione attiva utilizzando i cookie, dove viene settata la scadenza pari a quella del token.

È possibile effettuare il logout, che comporta nella semplice cancellazione dei cookie.

Viene definita un'ulteriore classe "AuthFactory", che fornisce un'istanza del tipo corretto dal quale autenticarsi.

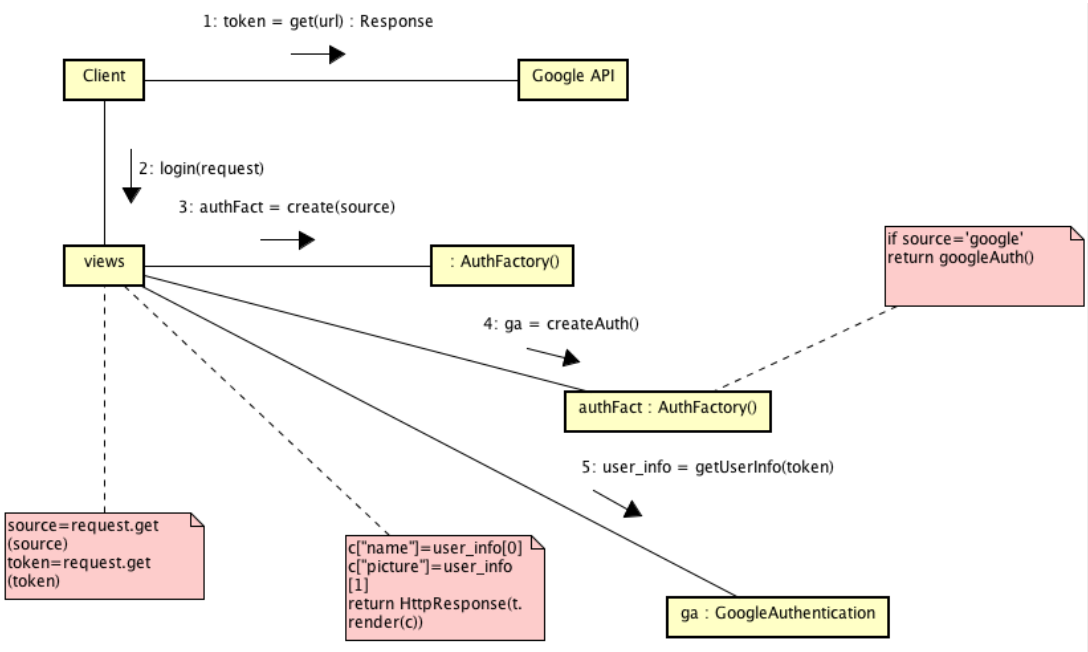

Figura 8. Diagramma di interazione dell'autenticazione attraverso Google

A differenza per Linkedin, poiché viene fornito il codice di autorizzazione, è necessario definire un'operazione in più fornita dalla classe "LinkedinAuthentication()", che permette di effettuare lo scambio per ottenere il token con una chiamata http, utilizzando le API REST messe a disposizione da Linkedin.

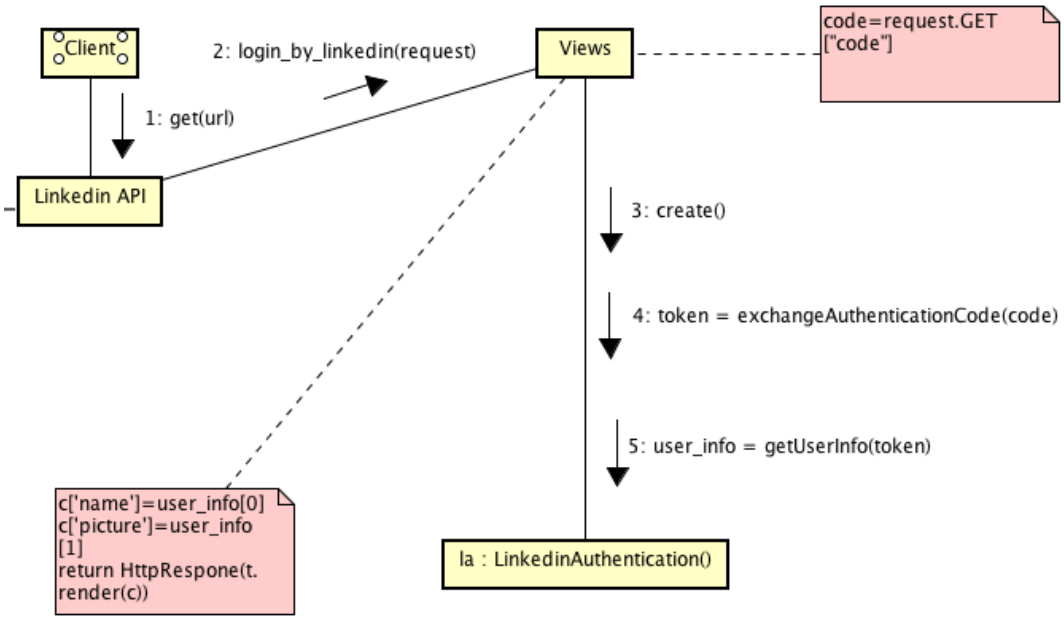

Figura 4. Diagramma di interazione dell'autenticazione attraverso Linkedin

6.2 Autorizzazione

In futuro l'applicazione sarà sottoposta ad una monetizzazione, quindi è necessario definire un sistema che permetta di web payment.

In particolare le serie si suddividono in serie a pagamento e gratuite in base alla sorgente.

L'idea è quella di sviluppare in futuro degli abbonamenti che l'utente può effettuare. Sarà possibile effettuare il download di tutte le sorgenti, viceversa non è possibile effettuare l'esportazione se non si possiede uno degli abbonamenti(descritti nel Capitolo 1.4).

SQLite viene utilizzato per le configurazioni di base e memorizzare le costanti utilizzate nell'applicazione.

Per sviluppare questa funzionalità è necessario rendere persistenti gli utenti che si autenticano nel sistema, con l'aggiunta degli abbonamenti che possiedono.

In fase di progettazione l'idea era di salvare gli utenti in SQLite, con una tabella dedicata agli abbonamenti.

Però è stata abbandonata sia per mantenere SQLite unico per le configurazioni e le costanti, sia perché è possibile che gli utenti in futuro siano un numero numeroso, e SQLite non è performante per numeri molto grandi.

Quindi è stata creata una *Collection* su MongoDB "users" e una tabella in SQLite, nella quale vengono salvate tutte le tipologie di abbonamento che è possibile effettuare, con riferimento ad un id univoco.

La *collection users* è composta da due campi: email, e *subscription*.

Subscription rappresenta una lista che contiene tutti gli abbonamenti effettuati, validi e scaduti.

Ogni elemento della lista subscription, contiene un campo "id" che si riferisce alla chiave della tabella in SQLite, che rappresenta l'abbonamento, ed un campo "*expire_in*" che indica la scadenza dell'abbonamento.

Gli abbonamenti si possono dividere in due tipologie (Descritte nel Capitolo 1.4): Abbonamenti temporali, Abbonamenti *pay-as-you-go.*

Quindi il campo "*expire_in*", rappresenta una data per un abbonamento temporale, altrimenti rappresenta un numero.

Un esempio di document all'interno di MongoDB è il seguente:

```
{
    "_id" : ObjectId("55eaf4ca68f62002bb61b1e6"),
    "email" : "marco.faretra.93@gmail.com",
    "subscription" : [
        {
            "id" : 2,
            "expire_in" : "2014-12-31"
        }]
}
```

Quindi ho creato una classe "User", dove all'interno ho inserito le operazioni base utilizzate, come per esempio il metodo "persist", che effettua la persistenza dell'utente su MongoDB utilizzando il modulo pymongo.

```
def __mongo_persist(self):
    db = self.ma.get_db()
    users = db.users

    if not self.__mongo_exists():
        log.info(self.email + " saving into MongoDb")
        dict_to_store = {"email": self.email, "subscription": []}
        users.remove({"email": self.email})
        users.insert_one(dict_to_store)
```

La classe User offre altri metodi ad esempio: un metodo che permette di accedere a tutti gli abbonamenti di un utente, un metodo che permette di rimuovere un utente dal database, un metodo che verifica se l'utente già esiste all'interno del database ed infine un metodo che nel caso di un abbonamento pay-as-you-go decrementa il valore della scadenza di 1.

Innanzitutto per implementare la funzionalità, ho effettuato il salvataggio nel database degli utenti, un utente viene inserito nel database se non è già presente quando si autentica, e viene inizializzato il campo "subscription" con una lista vuota. Per capire quali sorgenti sono a pagamento e quali gratuite, ed anche per facilitare l'aggiunta di ennuple in futuro, è stato necessario effettuare una tabella dedicata su SQLite, che avrà il campo "id" che sarà il nome della sorgente, ed il campo "is_free" che può avere valori "True" o "False".
Quindi nel metodo richiamato dal client, al momento in cui l'utente vuole esportare una serie storica, vengono effettuate le seguenti operazioni:

1. Prima si verifica se la serie storica è gratuita, in quel caso procedo con l'esportazione normalmente.
2. Se non è gratuita, effettuo un controllo, attraverso la classe "Auth", che verifica se è presente un abbonamento temporale o un abbonamento "pay-as-you-go", e se è presente procedo con l'esportazione, se è presente un abbonamento pay-as-you-go allora decremento di 1 il valore della scadenza.
   Altrimenti se non è presente nessun tipo di un abbonamento, o l'abbonamento è scaduto, ritorna alla pagina iniziale. In futuro questa parte sarà integrata con l'implementazione dei pagamenti attraverso paypal all'interno dell'applicazione.

7. Web Services REST

Il Capitolo 7 tratta della realizzazione di un'interfaccia REST per l'esportazione dei dati, tale interfaccia è composta da diversi vincoli e parametri da poter utilizzare.

7.1 Come funziona?

L'integrazione di questa funzionalità, comporta la possibilità di far utilizzare a sviluppatori i dati forniti da Glimpse e permette di migliorare la qualità del codice utilizzando un'unica piattaforma dove effettuare le operazioni principali.

L'interfaccia permette di scaricare e visualizzare le serie attraverso semplici URL http, in stile REST, specificando in formati standard tutte le opzioni già disponibili tramite la GUI.

L'unico metodo http supportato è il metodo GET.

Le risorse sono identificate dalle URL, un esempio è il seguente:

http://…/timeseries/<source>/<country>/<timeseries_id>

dove <source> è il nome della sorgente, <country> è il codice iso di un paese, che può essere anche ALL, <timeseries_id> è il codice della timeseries con cui può essere reperita.

La URL può avere anche un'altra forma:

http://…/timeseries/<source>/<timeseries_id>

In questo caso, viene evitato di inserire il parametro <country>, e facendo questa chiamata si intende implicitamente <country>=ALL.

Tale chiamata ritorna la *time series*, nel formato json.

Oltre la classica chiamata per estrarre e scaricare una *time series*, è possibile fare altre chiamate, come per esempio:

- http://…/timeseries/<source>/<country>: che ritorna sempre in formato json, l'elenco di tutte le serie storiche che hanno sorgente <source> e paese <country>.

- http://.../timeseries/<source>: che ritorna l'elenco di tutte le serie che hanno sorgente <source>.

- http://.../timeseries: che ritorna l'elenco di tutte le serie gestite da Glimpse.

Opzionalmente è possibile inserire dei query parameter nella query string, segue l'elenco di tutti i query parameter disponibili:

- *transformation = <transformation_type>*

Il valore di *<transformation_type>*, può essere una delle trasformate disponibili all'interno dell'applicazione, se è un parametro non accettabile, viene sollevata una Bad Request 400 spiegando il motivo.

Se non viene definito, come valore di default si intende "raw".

Ovviamente questo parametro può essere utilizzato solamente nella richiesta http completa di sorgente, paese e l'id della serie.

- *format = <format_name>*

Come per la *transformation*, può essere uno dei formati di esportazione disponibili all'interno dell'applicazione, se è un parametro non accettabile, solleva una Bad Request 400 spiegando il motivo.

Se non viene definito, come valore di default si intende "json".

Anch'esso solamente utilizzato nella richiesta completa di sorgente, paese ed id della serie.

- *metadata = <option>*

In questo caso il campo *<option>*, può valere "YES", "NO", o "ONLY", può essere applicato solamente se si richiede il formato json, altrimenti solleva un Bad Request 400 spiegando il motivo.

Il valore di default è NO.

Se si indica "YES", nel file json vengono ritornati sia i dati che i metadati, se si indica "NO" solamente i dati ed infine se si indica "ONLY" solamente i metadati.

- vintage = YYYY-MM-DD

Rappresenta la data in cui si richiede la serie storica, se non si specifica di default si intende la data odierna. Solleva un Bad Reqeust 400, se la data è scritta in formato errato.

Ritorna la serie storica con il vintage definito nel parametro.

- transformation_list

È un parametro senza valore, si può utilizzare soltanto insieme al parametro vintage, e ritorna l'elenco delle trasformate possibili al vintage dato es:[lag, log, raw].

Se utilizzato da solo o con altri parametri diversi da vintage, viene sollevata una Bad Request 400.

Anch'essa utilizzata nelle richieste http nel formato completo.

- vintage_list: è un parametro senza valore, si può utilizzare solamente da solo, e ritorna l'elenco dei vintage disponibili per la serie, es:['2012-02-01', '2015-08-21']. Se utilizzato con altri parametri solleva una Bad Request 400.
- query = <*sentence*>: è un'espressione da ricercare nel titolo/descrizione nella forma "Questa è un'espressione".

Questo parametro può essere utilizzato soltanto con le richieste http che forniscono un elenco di *time series*, mentre solleva un'eccezione Bad Request 400, se utilizzata con la richiesta http completa di sorgente, paese ed id della serie.

- *internal*: è un parametro senza valore, limita le serie ritornate a quelle scaricate in Glimpse. Può essere sempre usato, con qualsiasi richiesta.
- *prefix* = <sentence>: <sentence> è un'espressione da ricercare come prefisso nell'id della serie, nella forma "Questo è un prefisso", utilizzata per l'autocompletamento delle *time series*.

7.2 Sviluppo

Ogni qualvolta viene effettuata una chiamata http che prende in considerazione l'interfaccia rest, essa viene spedita ad un dispatcher chiamato rest, che appunto suddivide le richieste a seconda della tipologia della chiamata.

Quando viene inserita la URL completa per la richiesta di una *time series*, il dispatcher invoca un metodo che verifica prima se la serie è già presente su MongoDB, nel caso fosse presente ritorna direttamente la *time series* da Mongo, nel caso in cui non fosse presente, effettua la chiamata esterna ed il download e ritorna il risultato, rispettando i parametri.

Ho utilizzato questa tecnica, poiché risulta notevolmente più veloce la richiesta da MongoDB.

Quindi ho creato due metodi, uno per la gestione delle serie interne ed uno per la gestione delle serie esterne, quando viene effettuata una chiamata, innanzitutto verifica

che tutti i query parameter siano leciti, e successivamente effettua una chiamata per il metodo delle serie interne, ove verifico che la serie sia effettivamente interna, se lo è la ritorno rispettando i parametri, ed in caso contrario invoco il metodo per le serie esterne.

```
ts.load()
exporter = Exporter()
result = exporter.jmo(vintage, ts, transformation)

if json.loads(result)["data"] == []:
    if internal:
        return HttpResponseNotFound("The timeseries has not been downloaded")
    return get_external_timeseries(source, tscode, country, metadata, vintage,
transformation, format)

if vintage not in ts.get_available_vintage_dates():
    if internal:
        return HttpResponseNotFound("The timeseries has not been downloaded")
    return get_external_timeseries(source, tscode, country, metadata, vintage,
transformation, format)

if format != "JSON":
    return get_external_timeseries(source, tscode, country, metadata, vintage,
transformation, format)

log.info("Get Timeseries from Mongodb: "+str(tsname))
if metadata == "ONLY":
    temp = dict()
    temp["metadata"] = ts.metadata
    return HttpResponse(json.dumps(temp), content_type="application/json")
elif metadata == "YES":
    return HttpResponse(result, content_type="application/json")
else:
    temp = dict()
    temp["data"] = json.loads(result)["data"]
    return HttpResponse(json.dumps(temp), content_type="application/json")
```

Unica eccezione a riguardo è nel caso in cui viene specificato un formato diverso da json, allora in quel caso utilizzo le funzioni già esistenti per l'esportazione, ed effettuo quindi una chiamata esterna.

Nella chiamata esterna effettuo la chiamata per prendere i dati dalla sorgente esterna e li salvo all'interno di MongoDB.

```
ts = source_importer.get_ts_by_name(tsname.lower(), vintage)
exporter = Exporter()

if format != "JSON":
    ts.persist()
    if transformation is not None and transformation != "raw":
        get_transform_timeseries(ts, vintage, transformation)
    return export_in_format(tsname, vintage, transformation, format)

elif transformation is not None and transformation != "raw":
    get_transform_timeseries(ts, vintage, transformation)
    ts_response = exporter.jmo(vintage, ts, transformation)
else:
    ts.persist()
    ts_response = exporter.jmo(vintage, ts, transformation)
if metadata == "ONLY":
    temp = dict()
    temp["metadata"] = json.loads(ts_response)["metadata"]
    return HttpResponse(json.dumps(temp), content_type="application/json")
elif metadata == "YES":
    return HttpResponse(ts_response, content_type="application/json")
else:
```

```
temp = dict()
temp["data"] = json.loads(ts_response)["data"]
return HttpResponse(json.dumps(temp), content_type="application/json")
```

Differentemente, se effettuo chiamate di “ricerca”, ovvero non complete, vengono invocati altri metodi dal dispatcher.

In queste chiamate gli organi presi in considerazione sono ElasticSearch e MongoDB. Le tipologie di chiamate http da effettuare possono suddividersi in 3 casi:

- Per paese

- Per sorgente

- Tutte le serie storiche

Quindi ho sviluppato tre metodi separati ognuno che si occupa di una tipologia. Mongo viene preso in considerazione soltanto nel caso in cui è presente il query parameter “internal”, che intende quindi soltanto le serie storiche già scaricate in Glimpse.

Nel caso non fosse presente, vengono effettuate delle query su Elastic in base alla tipologia, che restituiscono i risultati desiderati in formato json, e vengono poi quindi mostrati.

Lo sviluppo di questa interfaccia è tornato utile, poiché il codice risulta suddiviso in modo migliore, ed è stato effettuato molto refactoring del codice.

In effetti molte funzionalità, come per esempio l’export, o quando vengono richieste le trasformazioni, sono state cambiate con richieste all’interfaccia rest. Inoltre è fondamentale per l’utilizzo dell’autocompletamento, ogni volta che si comincia a inserire all’interno dell’input, viene effettuata una richiesta all’interfaccia con il campo *prefix*, che mostra i migliori 10 risultati.

Conclusioni

L'obiettivo di questo tirocinio è stato l'ampliamento di un'applicazione web, orientato in particolare all'integrazione di serie storiche estratte da diverse sorgenti.

Durante questo periodo di tirocinio ho acquisito delle competenze tra le quali la conoscenza del linguaggio Python, l'utilizzo di MongoDB e la capacità di lavorare in team, in quanto il tirocinio è stato svolto all'interno dell'Università degli Studi Roma Tre, nel Laboratorio Basi di Dati, con il mio collega Antonio Martinelli, sotto la supervisione ed il supporto dell'Ing. Luigi Bellomarini.

L'applicazione inizialmente forniva la possibilità di estrarre dati da una sola sorgente, in particolare FRED e ben poche funzionalità.
Dopo l'attività di tirocinio sono state rese disponibili 9 sorgenti (descritte nel Capitolo 3 e 4), la possibilità di autenticarsi attraverso i social network (descritta nel Capitolo 6), la possibilità di utilizzare delle API REST (descritta nel Capitolo 7) fornite dall'applicazione ed infine la possibilità di effettuare delle ricerche per descrizione (descritta nel Capitolo 5), attraverso la piattaforma ElasticSearch.

In base a quanto detto, gli obiettivi prefissati sono stati ampiamente soddisfatti.

Bibliografia

Worldbank, API Documentation | Data, Worldbank API REST Documentation
http://data.worldbank.org/developers/api-overview
(Ultima consultazione 31/07/2015)

Eurostat, REST SDMX 2.1 – Eurostat, Eurostat REST SDMX Documentation
http://ec.europa.eu/eurostat/web/sdmx-web-services/rest-sdmx-2.1
(Ultima consultazione 20/08/2015)

Eurostat, Bulk Download, Eurostat Data
http://ec.europa.eu/eurostat/estat-navtree-portlet-prod/BulkDownloadListing
(Ultima consultazione 01/09/2015)

Opec, OPEC : OPEC Basket Price, Opec Data
http://www.opec.org/opec_web/en/data_graphs/40.htm
(Ultima consultazione 31/08/2015)

Quandl, Getting Started with the Quandl API, Quandl Documentation
https://www.quandl.com/blog/getting-started-with-the-quandl-api
(Ultima consultazione 10/09/2015)

Google, Using OAuth 2.0 to Access Google APIs | Google Identity Platform | Google Developers, Google Documentation
https://developers.google.com/identity/protocols/OAuth2
(Ultima consultazione 15/07/2015)

Linkedin, Sign In with LinkedIn | Documentation, Linkedin Documentation
https://developer.linkedin.com/docs/signin-with-linkedin
(Ultima consultazione 15/07/2015)

Amazon, Login with Amazon: Developer Guide for Websites, Amazon documentation

https://images-na.ssl-images-amazon.com/images/G/01/lwa/dev/docs/website-developer-guide._TTH_.pdf
(Ultima consultazione 15/07/2015)

Elastic, Document APIs, Elastic Search Documentation
https://www.elastic.co/guide/en/elasticsearch/reference/2.0/docs.html
(Ultima consultazione 25/08/2015)

Whoosh, Release notes Whoosh 2.7.0 documentation, Whoosh Documentation
http://whoosh.readthedocs.org/en/latest/releases/index.html
(Ultima consultazione 22/08/2015)

MongoDB, The MongoDB 3.0 Manual MongoDB Manual 3.0, MongoDB Documentation
https://docs.mongodb.org/manual/
(Ultima consultazione 15/07/2015)

Scrum, Scrum Guides, La guida all'utilizzo di Scrum
http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ITA.pdf
(Ultima consultazione 15/07/2015)

SDMX, Learning | SDMX Statistical Data and Metadata eXchange, La guida all'utilizzo del formato SDMX
http://sdmx.org/?page_id=2555
(Ultima consultazione 10/09/2015)

OAuth2.0, Using OAuth 2.0 to Access Google APIs, Introduzione ad OAuth2.0
https://developers.google.com/identity/protocols/OAuth2
(Ultima consultazione 25/07/2015)

Zniper, Zniper Blog - Django, Searching, Benchmark ElasticSearch/Whoosh/Solr
http://zniper.net/posts/django-searching/
(Ultima consultazione 10/10/2015)