Documentazione

PithyURL

Versione 1.0
Data di rilascio: 12/10/2015

PBDMNG a.a. 2014-2015

Realizzato da
Maiorano Federico 607951 ITPS f.maiorano2@studenti.uniba.it
Guerra Giuseppe 593277 ITPS guerraiuseppe1507@gmail.com
Lacalamita Pasquale 571399 ITPS lacalamita.pasquale@gmail.com

INDICE

1.0 Storia e motivazione del contesto

Un primo riferimento ad un servizio di **URL shortening** si può già trovare in un brevetto (http://1.usa.gov/1s7Jt7m) del 2000, scritto da Megiddo e McCurley di IBM. Brevetto registrato nel settembre 2000 ma emesso solo nell'ottobre 2005.

Tuttavia il primo servizio di URL shortening è considerato **TinyURL**; lanciato nel 2002, il suo successo ha determinato la nascita di almeno 100 siti web simili, anche se la maggior parte di questi fungono esclusivamente da alternative per il dominio.

Il successo degli URL shortener si deve soprattutto a Twitter; infatti, dato il limite di lunghezza di un tweet impostato a 140 caratteri, è essenziale ottimizzare lo spazio accorciando tutti quei link eccessivamente lunghi.
Twitter inizialmente trasformava automaticamente long URL utilizzando TinyURL, dal 2009 però **Bit.ly** è divenuto l'URL shortener di default. A causa di ciò, dal 2009 Bit.ly ha sostituito TinyURL, divenendo il servizio di URL shortening con il maggior volume di traffico online.

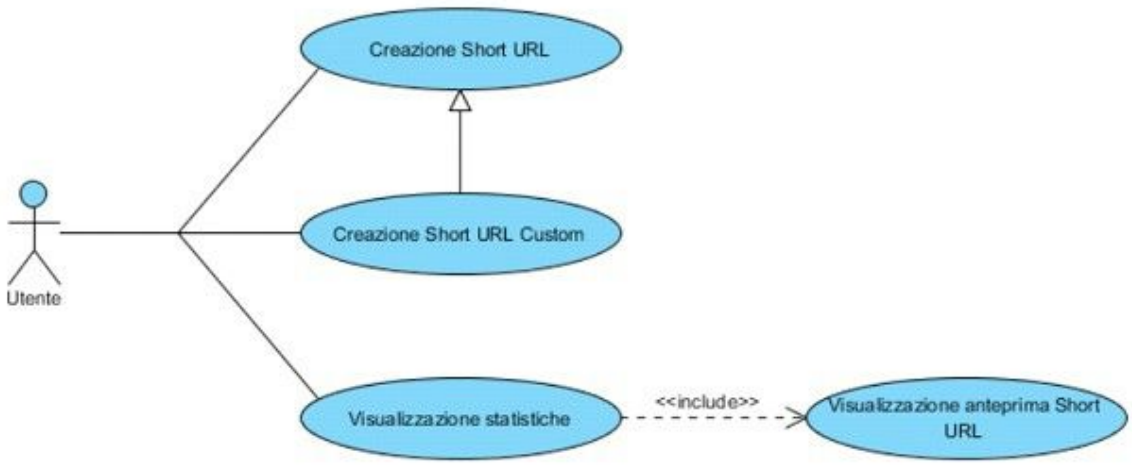
Nel dicembre 2009 Google annunciò un servizio di URL shortening all'indirizzo goo.gl, che originariamente era disponibile solo per l'uso su alcuni prodotti di Google, quali Google Toolbar e FeedBurner, e estensioni di Google Chrome. In seguito Google presentò anche un URL Shortener per YouTube (youtu.be). **Goo.gl**, come anche altri suoi simili, fornisce statistiche sul link e un QR code per l'accesso da dispositivi mobili.

Il maggiore vantaggio di questo tipo di servizi è che uno short link può essere facilmente comunicato, ed il suo utilizzo in testi o in messaggi su social network migliora la leggibilità. Tuttavia, in molti sfruttano il fatto dell'oscuramento del reale indirizzo di destinazione per spamming o per dirigere l'utente su siti malevoli.
Per ovviare a questo problema molti servizi di URL shortening offrono la possibilità di una preview del link abbreviato, in modo da aver una idea di dove lo short URL ci sta per reindirizzare.

2.0 Analisi delle funzionalità

Il sistema realizzato permette di:

- Creare uno short URL a partire da un URL standard (con controllo del URL inserito tramite espressioni regolari che consentono di rilevare URL errati);
- Creare uno short URL customizzato (con controllo su parole non ammesse);
- Visualizzazione di statistiche relative ad uno short URL creato;
- Preview di uno short URL (inclusa nella sezione statistiche).



3.0 Frequenza delle funzionalità

Le funzionalità descritte sopra hanno diversi livelli di frequenza d'uso; nel nostro servizio sicuramente la funzionalità più usata di frequente sarà la visita di uno short URL, infatti uno short URL potrebbe accumulare milioni di visualizzazioni dal momento in cui viene creato.
La creazione di uno short URL è frequente ma in misura minore al precedente; nei social network gli short URL sono molto usati per migliorare la leggibilità dei messaggi, vengono spesso condivisi (e quindi creati), ma sicuramente saranno cliccati in misura maggiore.
Infine le operazioni di **anteprima** e **visualizzazione delle statistiche** sono le operazioni meno frequenti, ma c'è da fare una considerazione sulla tipologia di utente che usa tale servizio, in quanto per un utente standard potrebbero non essere necessarie e poco utilizzate, ma per le aziende il discorso cambia in quanto possono analizzare le statistiche delle visite per studi di mercato dei propri prodotti.

4.0 Analisi dei costi delle operazioni

Di seguito verranno analizzati i costi delle operazioni in termini di letture e scritture di dati nel database.

La **creazione** di short URL a partire da un long URL richiede la lettura del database per verificare che non sia stato già generato uno short URL per quello stesso long URL. Qualora non sia esistente uno short URL per quel long URL, ne viene generato uno nuovo e si procede alla scrittura nel database di long URL, short URL, data di creazione, visite totali e uniche (entrambe inizializzate a zero), e un array vuoto che verrà man mano riempito e conterrà il numero di click proveniente da ogni nazione. Al contrario, se è già presente in database uno short URL relativo al long URL di partenza, viene restituito all'utente lo short URL precedentemente creato.

La **visita** di uno short URL comporta una lettura per ricavare il long URL e un update per modificare i campi delle visite totali e delle visite uniche, aggiornando anche il numero di click della nazione da cui lo short URL viene visitato, del documento con lo short URL richiesto.

La **visualizzazione** statistiche e anteprima di uno short URL comporta la sola lettura del documento dello short URL richiesto dal database.

La **creazione** di uno short URL customizzato comporta gli stessi costi della creazione di uno short URL standard.

5.0 Modello di dati

Un modello dei dati è un insieme di concetti utilizzati per organizzare una base di dati e descriverne la struttura in modo che essa risulti comprensibile ad un elaboratore. Ne esistono di diverse tipologie tra quelli adottati dai database NoSQL, ciascuno consente differenti paradigmi di interazione tra applicazione e database. Per il sistema di URL shortening che si è realizzato si è utilizzato come modello dei dati il **document**.

I database NoSQL che sono realizzati secondo questo modello memorizzano i dati come un documento con campi come attributi. Ogni documento ha un _id univoco e al suo interno si possono definire diversi campi direttamente indicizzabili. I document possono essere di formati diversi (XML, JSON, BSON), sono autodescrittivi, la loro struttura dati è di tipo gerarchico ad albero (BTREE) e interrogati utilizzando query JavaScript. I documenti hanno differenze nei loro attributi ma appartengono alla stessa collezione, diversamente da quanto accade nei database relazionali in cui le colonne memorizzano lo stesso tipo di valori o null.
I dati dispongono di uno schema flessibile, questo aiuta la corrispondenza del documento agli oggetti. Le relazioni possono essere rappresentate come riferimenti o documenti incorporati (references o embedded documents).

La scelta di questo tipo di modello dei dati è stata maggiormente influenzata dalla flessibilità dello schema che ci permette di memorizzare statistiche in modo più efficiente, al contrario di un key-value che ci avrebbe complicato un po' la strutturazione in quanto è molto free-schema, ma anche dalle performance in quanto le query sono possibili anche sui singoli campi di un documento.
Nel database, quindi, ogni document contiene le info di uno short URL e le sue statistiche:

```
document {
        "long" : String ,
        "short" : String ,
        "tot_visits" : int ,
        "unique_visits " : int ,
        "create_date" : Date ,
        "countries" : [
            { "name" : String ,
            "visits" : int }*
        ]
}
```

Dove con Tot_visits si intende il numero di click assoluto, mentre con Unique_visits il numero di visite singole distinte per indirizzo IP (viene infatti contato solo il primo click per ogni IP). Invece Countries è un array contenente nome e numero di visite di ogni paese da cui lo short URL viene cliccato.

6.0 Scelta del database NoSQL

Deve essere realizzato un **URL shortener**, un servizio che, dato un link in input, ne restituisca un altro abbreviato, generato in modo casuale oppure in modo personalizzato a partire da una stringa inserita dall'utente. Si deve consentire anche di visualizzare alcune statistiche relative ad uno short URL generato dal sistema, relative al numero di visite e la loro provenienza geografica in base all'IP.

Obiettivo primario sarà quindi quello di memorizzare la corrispondenza tra link in input (long URL) e link abbreviati generati dal sistema (short URL), in modo che ogni qualvolta un utente cliccherà su uno short URL verrà reindirizzato al long URL corrispondente. Oltre a tener memoria della corrispondenza long-short URL, si dovranno anche registrare ulteriori informazioni, da cui poter poi elaborare delle statistiche.

Si è deciso quindi che il miglior metodo per poter aggregare questi dati, fosse quello di unirli in un documento, ci siamo perciò indirizzati verso un database NoSQL Document-oriented.

La scelta del database SQL è stata effettuata considerando anche il **CAP Theorem**, secondo cui è impossibile che un sistema distribuito soddisfi contemporaneamente tutte e tre le caratteristiche di **consistenza**, **disponibilità** e **tolleranza della partizione**, ma al massimo 2. Sono state considerate di maggior rilevanza le caratteristiche di tolleranza della partizione, che garantisce il servizio anche qualora una partizione non comunichi con le altre, e di consistenza, in modo che i dati rimangano consistenti dopo l'esecuzione di una scrittura o di un update.

MongoDB è il database scelto, visto che appartiene alla categoria dei database NoSQL orientati ai documenti e fa parte di quei database che soddisfano contemporaneamente le caratteristiche evidenziate in precedenza; MongoDB risiede dunque su una definizione **CP** (Cap Theorem), ovvero assicura che, pur potendo essere qualche dato inaccessibile, il resto sia accurato e consistente.

MongoDB è il più popolare database document-oriented e NoSQL, con più di 10 millioni di download, migliaia di clienti e più di 1000 technology e service partner.
I documenti di MongoDB sono in stile JSON composti da campi chiave-valore, come ad esempio: {"name": "mongo", "age": 5}. MongoDB memorizza i documenti sul disco in formato BSON, dove BSON è la rappresentazione binaria di documenti JSON.

MongoDB garantisce **performance elevate**, in particolare:

- il supporto per i modelli di dati embedded riduce le attività di I/O su database
- gli indici garantiscono query più veloci e possono includere chiavi da documenti embedded e array.

Fornisce **alta disponibilità** attraverso una funzione di replicazione (Replica Set) che fornisce un failover automatico e ridondanza dei dati. Un Replica Set è un gruppo di MongoDB server che mantengono lo stesso data set, fornendo ridondanza e incrementando la disponibilità dei dati.
Infine MongoDB assicura **scalabilità orizzontale** tramite sharding automatico, distribuendo i dati attraverso cluster, e Replica Set, che forniscono letture eventually-consistent.

Da un report pubblicato dalla United Software Associates (USAIN), dove vengono messe a confronto le prestazioni di tre database NoSQL, Cassandra, Couchbase e MongoDB, quest'ultimo è stato quello che si è distinto nei test prestazionali (benchmark) effettuati dai ricercatori. I database sono stati messi alla prova dallo Yahoo! cloud standard benchmark (YCSB) con l'intenzione di accertarne l'effettiva durabilità; il metro di giudizio verte quindi sull'idea che le applicazioni debbano focalizzarsi più sulla durabilità che sulle performance, non tollerando perdite di dati.
Di seguito i risultati dei test, considerando che tutti i valori indicano il numero di operazioni al secondo:

- **Test 1** – ***throughput optimised***, rispettivamente 50% read – 50% update e 95% read – 5% update
  MongoDB: 160,719 – 196,498
  Cassandra: 134,839 – 144,455
  Couchbase: 106,638 – 187,798

- **Test 2** – ***durability optimised***, rispettivamente risultati workload A e B
  MongoDB: 31,864 – 114,455
  Cassandra: 6,289 – 54,864
  Couchbase: 1,236 – 18,201

- **Test 3** – ***balanced,*** rispettivamente risultati workload A e B
  MongoDB: 114,245 – 183,152
  Cassandra: 77,676 – 71,643
  Couchbase: (nessuna configurazione equivalente disponibile)

Come si evince dai risultati MongoDB risulta il migliore in termini di **durabilità**. Anche per questo la nostra scelta è stata indirizzata verso questo database, dato che la consideriamo abbastanza rilevante in un servizio di URL shortening. Si desidera infatti che rimanga "intatta" a lungo l'associazione tra uno short URL e il corrispettivo long URL, ovvero che cliccando su uno short URL, anche a distanza di tempo dalla sua creazione, si venga correttamente reindirizzati, anche a scapito del tempo.

Fonti:

- Introduction to MongoDB [https://docs.mongodb.org/manual/core/introduction/] ;
- Il database NoSQL più performante? È MongoDB
  [http://www.hostingtalk.it/il-database-nosql-piu-performante-mongodb/] .

7.0 Implementazione

7.1 DAO

È stata realizzata un'interfaccia per il Data Access Object (**IDAO**), contenente tutti i metodi necessari per realizzare le varie funzionalità del sistema, e una sua implementazione, **MongoDBDAO** (comunica con il server tramite una libreria com.mongodb).

7.2 Utils

Nel package Util oltre a **geoLocation** e **wordChecker** sono presenti altre classi di supporto.

7.2.1 CookiesHandler

permette di memorizzare nei cookies dell'utente la lista degli short URL visitati, tale espediente è stato utile per tenere aggiornato in modo consono la statistica unique_visits. Quindi incrementare la statistica solo la prima volta che l'utente visita uno short URL.

7.2.2 LongUrlValidator

è responsabile del controllo del long URL inserito dall'utente. Come primo passo viene fixato auto-completandolo creando un URL standard. Poi viene convalidato tramite una espressione regolare (**URL_REGEX**).

7.2.3 ShortiLinkGenerator

è la classe responsabile della creazione dello short URL verificando, sia nel caso sia stato creato dall'utente (custom URL) o dalla generazione casuale della classe, che sia univoco facendo un controllo nel database.

7.2.4 ShortLinkValidator

verifica che lo short URL personalizzato dell'utente sia ammesso.

7.2.5 JsonTransformer

permette di tradurre un JSON in un oggetto di tipo String.

7.2.6 Geo Location By IP

Tale classe è responsabile sia della geo-localizzazione di un IP, sia della gestione dell'oggetto grafico (tramite librerie *) presente nella sezione del servizio URL Stats che visualizza le statistiche di uno short URL. In tale sezione oltre alle statistiche come le visite totali e uniche, è presente una mappa del mondo in cui è possibile visualizzare quante visite sono state fatte da ogni paese.
***Rif.** Librerie usate (http://dev.maxmind.com/geoip/#GeoIP2)

7.2.7 Word Checker

La classe **WordChecker** agisce come filtro nell'inserimento del custom URL da parte dell'utente, e filtra le parole non ammesse, reputate non idonee. Viene applicato su diversi tipi di linguaggi, in particolare le 11 lingue principali in Europa.

7.3 Server

Il server è stato realizzato con l'ausilio del framework **Spark java**.

7.3.1 Bootstrap

ha la responsabilità principale di settare le varie configurazioni del server e tramite la classe Resource settare gli endPoint e catturare le varie richieste fatte dal client ed eseguire i servizi richiesti. Inoltre tale classe cattura le varie eccezioni generate dall'esecuzione dei servizi del sistema:

- **UndesirableWordException**
- **BadURLFormatException**
- **ShortUrlDuplicatedException**
- **ShortURLMaxLenghtReachedException**
- **ShortUrlNotFoundException.**

7.4 Client

Il **client** è stato realizzato tramite il framework **AngularJS**. Le principali funzionalità sono state realizzate nei file .js main e stats, nel main sono gestiste le richieste al server per la creazione di uno shortURL e in stats la gestione delle statistiche di uno shortURL che si vuole ispezionare.
Il client è formato da due principali file html, index e 404. La 404 è la pagina a cui si viene reindirizzati in caso il server non trovi lo short URL richiesto o la pagina a cui dovrebbe reindirizzare.
La index è la pagina principale a cui si può accedere a tutte le funzionalità del servizio.
Per la grafica delle statistiche delle visite nel mondo abbiamo utilizzato la libreria **jQuery Mapael**. (http://www.vincentbroute.fr/mapael/).
Per la grafica della pagina web si è utilizzato il CSS di **Twitter Bootstrap**.

8.0 TEST

I seguenti test sono stati eseguiti per mezzo del framework JUnit (Contrassegnati con un pallino i metodi testati).

8.1 DataAccess

8.1.1MongoDBDAOTest

- **createNewLsUrl**

Viene controllato se un nuovo short url venga salvato correttamente nel database che sia custom o meno.
Viene testato che venga restituito uno short url già esistente se è già stato creato uno per uno stesso long url.
Inoltre viene testato che vengano lanciate correttamente le eccazioni legate al metodo quali BadURLFormatException, ShortUrlDuplicatedException e UndesirableWordException.
N.B in questo caso di test viene testato automaticamente anche il metodo **getLsUrl.**

- **visitLsUrl**

Viene testato che dopo una visita di uno short url vengano incrementati i corretti contatori e aggiornati i corretti campi nel database.

8.2 Util

8.2.1 WordCheckerTest

- **isUndesiderable**

Viene testato che il metodo restituisca il corretto booleano a seconda che la parola in input sia consentita o meno.

8.2.2 CookiesHandlerTest

Questo test viene eseguito avviando il server temporanemante con due route di test abilitate di cui una per il salvataggio dello short url visitato nei cookies ed uno per il recupero di informazioni dai cookies.

- **handleVisit**

Viene testato che un cookie venga aggiornato o meno a seconda che sia stato già visitato o no e che quando viene visitato uno short il cookie venga aggiornato correttamente.

8.2.3 LongUrlValidatorTest

- **validate**

Verifica che la regex implementata dal metodo accetti tutti i tipi di url che ci si aspetti.

8.2.4 ShortUrlValidatorTest

- **validate**

Verifica che la regex implementata dal metodo accetti tutti i tipi di short url che ci si aspetti.

8.2.5 ShortLinkGeneratorTest

- **generaLink**

Verifica che lo short url random generato sia nel formato atteso.