URLShortnter

Paolo Cattaneo

Progetto Tecnico per Deltatre

# Requisito

Si richiede di sviluppare un sistema autoconsistente URLShortner. Il sistema deve poter accorciare gli URL inseriti dall’utente, salvarli in un sistema di dati persistente in modo che siano disponibili anche successivamente, e infine ridirigere su richiesta l’utente nel momento in cui inserisce il link accorciato.

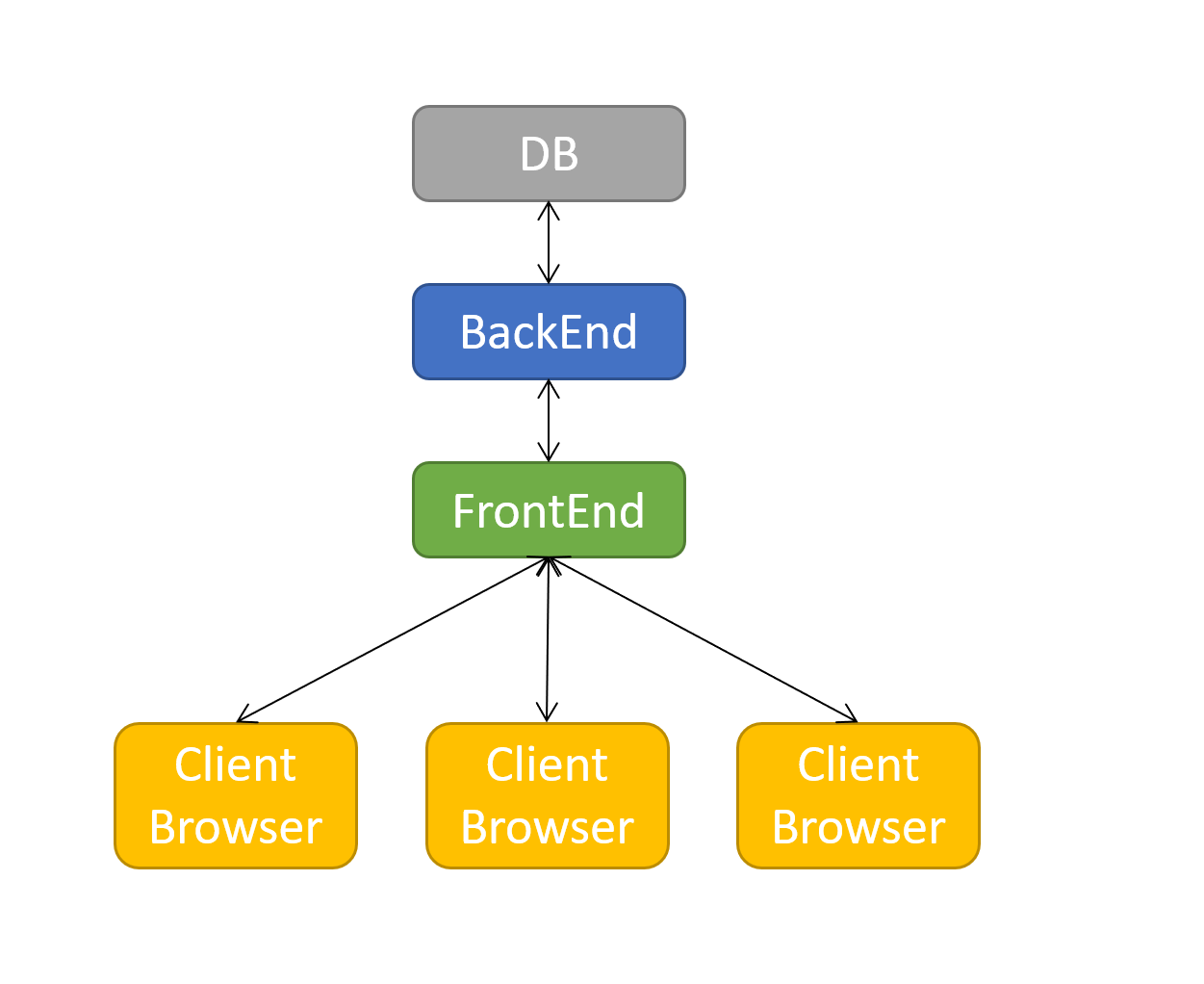
I temi tecnici più interessanti da notare nel requisito sono:

1. Il link contiene una scadenza (non è salvato all’infinito), associata automaticamente in fase di generazione. Il documento non specifica se questa scadenza debba essere configurabile o selezionabile dall’utente (ad esempio con un selettore di scadenza “1 mese-3 mesi-6 mesi-Mai”). Pertanto ho deciso di utilizzare una scadenzaa fissa di 3 mesi.
2. La stringa unica è specifcatamente “short as needed” e “composed by simple character”, pertanto ho applicato delle regole nella costruzione della stringa, e mi sono allontanato dall’idea originale di utilizzare dei Guid. Maggiori dettagli nel capitolo relativo.

Il requisito non indica tassi di arrivo particolari, o tempi di risposta minimi richiesti, pertanto ho cercato di utilizzare il buonsenso nella definizione della soluzione.

# Architettura

Il sistema è Web-based ed è costituito da vari componenti, illustrati nello schema sottostante.



L’architettura non entra volutamente nel dettaglio implementativo dei singoli moduli, che si possono considerare come placeholder di funzioni e ruoli all’interno dello schema.

## Front End

Come richiesto il modulo FrontEnd consiste in WebApp, sviluppata in Angular, molto semplice. La pagina contiene soltanto

1. Una casella di testo
2. Un pulsante
3. Una sezione di testo in cui viene scritto l’URL accorciato dal sistema

La WebApp effettua chiamate REST al sistema BackEnd che restituisce il dato processato. Nessuna funzionalità è portata sul FrontEnd al netto dell’interazione con l’utente.

La WebApp contiene anche una pagina statica che viene mostrata all’utente nel caso in cui la pagina non può essere

## BackEnd

Il modulo BackEnd consiste in un’applicazione ASP.NET che espone API per l’interazione con il database.

Le funzionalità necessarie sono fondamentalmente 2:

1. Inserimento di nuovi URL accorciati nel sistema (POST)
2. Ritrovamento di URL già calcolati (GET).

Fare riferimento alla documentazione Swagger esportata nel file TODO.

## DB

Il modulo Database astrae il concetto di modulo in cui vengono salvati gli URL accorciati e dal quale vengono successivamente estratti gli URL già calcolati.

Gli approcci sono molteplici:

1. Database relazionali (es: MySQL)
2. Database noSql (es: MongoDB)
3. Databasse in memoria (es: Redis)

TODO usato MongoDB?

# L’algoritmo di codifica/decodifica

L’idea originale era quella di utilizzare dei Guid, i quali garantiscono univocità, ma sono troppo lunghi e verbosi per la richiesta originale. Immaginando che il problema non fosse stato posto per la prima volta a me, ho cercato della letteratura a riguardo.

Tra le varie soluzioni proposte ho adattato al mio caso una soluzione che prevede il seguente flusso.

#### Inserimento (nuovo URL accorciato)

1. Quando il sistema riceve una richiesta inserisce un oggetto “url” nel database, ottenendo un id autoincrementale. L’univocità dell’id autoincrementale è garantita dal DBMS.
2. L’id autoincrementale viene trasformato in una breve stringa tramite una funzione biunivoca. Questa stringa costituisce il campo “shortened” dell’oggetto “url”.
3. L’oggetto “url” viene salvato per intero nel database

#### Recupero di un URL accorciato

1. Il sistema riceve una breve stringa (url accorciata)
2. Utilizzando la funzione biunivoca dalla stringa recupera l’id autoincrementale originario
3. L’oggetto “url” viene recuperato dal DBMS tramite accesso per chiave

Pro:

* La stringa è garantita in modo univoco dall’id autoincrementale
* Non è necessario verificare sul database che un url accorciato sia già stato salvato ed utilizzato per altri url

Contro:

* I DBMS NoSQL (es: MongoDB) o in memoria (es: Redis) non supportano nativamente l’id autoincrementale. Passare a database relazionali solo per questo motivo non mi è parso consono, per cui ho trovato una procedura per “forzare” un \_id autoincrementale in MongoDB.
* Conoscendo l’algoritmo, o immaginandolo, è possibile scorrere artificialmente la tabella degli url, incrementando o decrementando l’ultimo carattere della stringa. Questo non è specificato come criticità nel requisito, ma preferisco comunque annotarlo.
* A causa di limitazioni interne tecniche del linguaggio di programmazione il sistema è limitato a valori di id inclusi in un intero a 32 bit, quindi può memorizzare fino a 2 miliardi di url.

Riporto per completezza altre possibili soluzioni che sono state scartate.

1. L’idea di codificare (utilizzando per esempio un algoritmo MD5) l’URL ricevuto in ingresso. Questo approccio è stato scartato perché
   1. Più utenti che codificano lo stesso URL ottengono lo stesso identificativo. Nonostante non fosse specificato nell’argomento l’ho ritenuto non adatto
   2. Per evitare conflitti (anche tra URL diversi, causa la non unicità dell’hash) sarebbe stato necessario appendere all’URL una chiave univoca. Avendo a disposizione degli utenti loggati si poteva pensare di appendere la chiave dell’utente, ma sarebbe stato comunque necessario effettuare un ulteriore controllo a database
   3. Questo approccio tuttavia va riconsiderato se il limite di 2 miliardi di URL memorizzati è stringente per il caso d’uso
2. L’ida di utilizzare un generatore di chiavi offline, il quale genera continuamente nuove chiavi univoche che il motore principale poi utilizza e consegna agli utenti. Questa idea è stata scartata nonostante l’ottimalità della stessa per evitare di aggiungere entropia al sistema, che volevo mantenere quanto possibile semplice.

# Progetto BackEnd: UrlShortner

Il progetto di Backend è contenuto nella soluzione UrlShortner.sln, che è a sua volta suddivisa ed organizzata in progetti.

## UrlShortner.Core

Progetto contenente il Core del Backend, suddiviso in cartelle secondo il design pattern che organizza il contenuto di un progetto Core in tre sottocategorie: Mode, DAL (Data Access Layer), BL (Business Logic).

Solitamente è consigliato assegnare ad ognuna di queste categorie un progetto specifico, ma data la relativa semplicità del modulo ho preferito riaccorpare il tutto in un unico progetto “Core”, pur tenendo i namespace differenziati.

## UrlShortner.ConsoleTest

Progetto console di test che ho utilizzato per i primi test sull’algoritmo di chiave e in generale per avere un ambiente dove lanciare e debuggare singoli componenti di codice.

In un progetto “reale” questo *non* è l’approccio consigliato, poiché è meglio approntare un progetto di Unit Test ben strutturato, ma sempre per il beneficio della rapidità e del *quick & dirty* ho utilizzato questo approccio.

## UrlShortner.API

Progetto di API che ospita gli endpoint per le API necessarie per le funzionalità richieste.