

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.

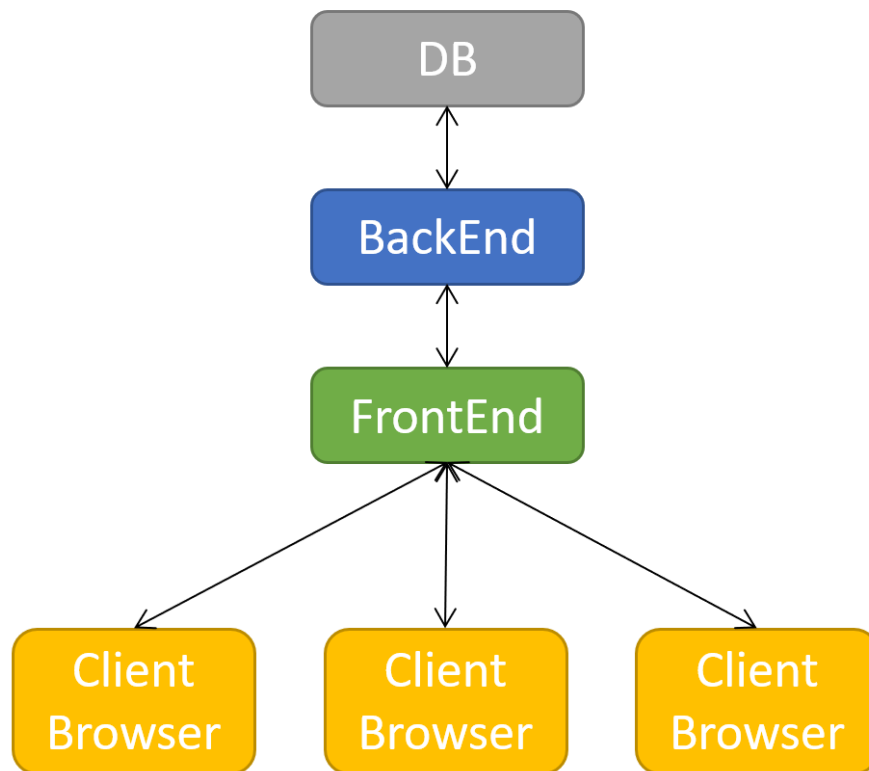
I temi tecnici più interessanti da notare nel requisito sono:

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 scadenza fissa.
2. La stringa unica è specificatamente "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 nella sezione relativa.

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.

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 (chiave) 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
 - a. Più utenti che codificano lo stesso URL ottengono lo stesso identificativo. Nonostante non fosse specificato nell'argomento l'ho ritenuto non adatto
 - b. 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
 - c. Questo approccio tuttavia va riconsiderato se il limite di 2 miliardi di URL memorizzati (int.MaxValue) è stringente per il caso d'uso

2. L'idea 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.

DATABASE

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.

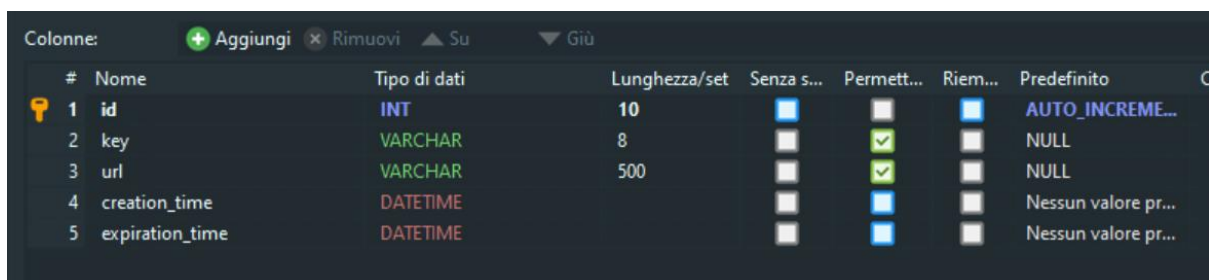
Gli approcci candidati sono molteplici:

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

L'idea originale era quella di utilizzare un database noSQL (MongoDB) per il salvataggio degli URL accorciati.

Il problema è che, in base all'algoritmo di codifica/decodifica scelto, è necessario utilizzare un database che supporti gli id autoincrementali. MongoDB purtroppo non li supporta nativamente, e li supporta con un workaround basato sui trigger, disponibile soltanto nella versione cloud.

Volendo mantenere una soluzione on premise, sono passato ad una tecnologia a me più nota, MySQL. Il modello è semplice e consiste in una sola tabella *url*.



Colonne:							
#	Nome	Tipo di dati	Lunghezza/set	Senza s...	Permett...	Riem...	Predefinito
1	id	INT	10	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREME...
2	key	VARCHAR	8	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
3	url	VARCHAR	500	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
4	creation_time	DATETIME		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nessun valore pr...
5	expiration_time	DATETIME		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nessun valore pr...

Per l'accesso al database ho utilizzato Entity Framework Core, con un approccio Database-First, generando le classi C# necessarie con lo script consultabile nel repository consegnato (*urlshort-db-scaffold.ps1*).

Nel repository principale ho consegnato anche un dump del database di test, esportato nel file (*urlshortner_db.sql*) in modo che si possa consultare anche direttamente.

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: Model, 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 riaccorpate il tutto in un unico progetto “Core”, pur tenendo i namespace differenziati.

- Model: classi DTO
- DAL: accesso al database
- BL: metodi di business logic, utilities

Nonostante il modello di dati abbia solo un oggetto e non cambia tra database e modello restituito dalle API ho comunque mantenuto un pattern che li tiene separati. Il mapping tra il layer Database e il layer Business Logic è effettuato utilizzando la libreria Automapper.

URLSHORTNER.CONSOLE

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.

Lo swagger di documentazione è reperibile lanciando le API e accedendo a “.../docs”.

UrlShortner.API ^{v1} OAS3

/swagger/v1/swagger.json

Redirect

GET `/key` Redirect to the URL corresponding to the key.

Urls

GET `/api/urls/key` Get the a shortned URL corresponding to the given key.

DELETE `/api/urls/key` Delete an URL from the database.

POST `/api/urls` Encode a new URL into the database.

Schemas

UrlShort >

PROGETTO FRONTEND: URLSHORTNER-WEBAPP

Il progetto FrontEnd è sviluppato in Angular, come richiesto esplicitamente.

L'applicazione consiste in una singola pagina che può contenere vari componenti e gestisce tramite un cliente HTTP le richieste al server. Le funzionalità riservate all'app sono di sola creazione dell'URL abbreviato invocando la POST alle API.

SHORTENCOMPONENT

Contiene le funzionalità di accorciamento dell'URL inserito dall'utente. L'utente che atterra sulla pagina principale viene ridirezionato direttamente su questo componente che contiene la funzionalità principale.



URL Shortner WebApp

Shorten your URL

NOTFOUNDCOMPONENT

Viene raggiunto se l'utente cerca di accedere ad un link che non viene trovato dalle API sul database. La gestione di "non trovato" e "scaduto" è sovrapposta e non viene distinta a livello di FrontEnd.



URL Shortner WebApp

Link not found or expired

REDIRECTCOMPONENT

Questo componente viene raggiunto ogni volta che un utente inserisce un possibile link accorciato. Si occupa di invocare l'API che richiede l'URL e, in caso affermativo (sia di presenza che di non-scadenza), ridirige l'utente all'URL richiesto.

MIGLIORIE

Il lavoro svolto è un punto di partenza a cui si possono portare migliorie.

1. L'algoritmo di chiavi non è sostenibile all'infinito e si basa su un id autoincrementale fornito dal DBMS. Il sistema più corretto, che non è stato implementato poiché richiedeva un lavoro che ho stimato essere maggiore dei giorni assegnati, è quello del "generatore di chiavi" offline.
2. Cambiando l'algoritmo di chiavi è possibile anche passare da un database relazionale (totalmente ingiustificato data l'assenza di altre tabelle e quindi anche di relazioni) ad un database noSQL più efficiente e adatto al tema.
3. Il FrontEnd è scarso e va migliorato soprattutto a livello grafico. La gestione di "link non presente" e "link scaduta" può essere gestita in modo che l'utente sia in grado di distinguere i due casi.