Go: creare un URL shortener

Un URL shortener è uno strumento che permette di ridurre la lunghezza di un URL mantenendone la funzionalità originale. È particolarmente utile per condividere link lunghi su piattaforme che limitano il numero di caratteri o semplicemente per migliorare l'estetica del link. In questo articolo, vedremo come implementare un semplice URL shortener utilizzando il linguaggio di programmazione Go.

Per seguire questa guida, è necessario avere:

- Una conoscenza di base di Go.
- Go installato sul proprio sistema.
- Un ambiente di sviluppo configurato (ad esempio VS Code, GoLand, ecc.).

Prima di iniziare a scrivere il codice, è importante capire come funzionerà il nostro URL shortener. Il sistema dovrà:

1. **Ricevere un URL lungo** tramite una richiesta HTTP POST.
2. **Generare un identificatore univoco** per l'URL.
3. **Memorizzare l'associazione** tra l'identificatore e l'URL originale in un database.
4. **Restituire un URL accorciato** che punta al nostro servizio.
5. **Reindirizzare** l'utente all'URL originale quando visita l'URL accorciato.

Iniziamo creando una nuova cartella per il progetto e configurando il nostro ambiente Go.

```
mkdir url-shortener
cd url-shortener
go mod init url-shortener
```

Creiamo un semplice server HTTP che gestirà le richieste per accorciare e risolvere gli URL.

```
package main

import (
        "fmt"
        "log"
        "net/http"
)

func main() {
        http.HandleFunc("/", HomeHandler)
        http.HandleFunc("/shorten", ShortenHandler)
        http.HandleFunc("/resolve/", ResolveHandler)

        fmt.Println("Starting server on :8080")
        log.Fatal(http.ListenAndServe(":8080", nil))
}

func HomeHandler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "Welcome to the URL Shortener!")
}
```

Per accorciare un URL, abbiamo bisogno di generare un identificatore unico. Una semplice implementazione potrebbe essere generare una stringa casuale. Tuttavia, per semplicità e prevedibilità, possiamo utilizzare un contatore incrementale e convertirlo in una stringa breve.

```
import (
        "encoding/base64"
        "sync/atomic"
)

var idCounter uint64

func generateShortID() string {
        id := atomic.AddUint64(&idCounter, 1)
        return base64.URLEncoding.EncodeToString([]byte(fmt.Sprintf("%d", id)))
}
```

Ora che possiamo generare un identificatore, creiamo l'handler per accorciare gli URL. Questo endpoint riceverà un URL lungo tramite una richiesta POST e restituirà l'URL accorciato.

```
import (
        "encoding/json"
        "io"
        "net/http"
)

var urlStore = make(map[string]string)

type ShortenRequest struct {
        URL string `json:"url"`
}

type ShortenResponse struct {
        ShortURL string `json:"short_url"`
}

func ShortenHandler(w http.ResponseWriter, r *http.Request) {
        var req ShortenRequest
        body, _ := io.ReadAll(r.Body)
        json.Unmarshal(body, &req)

        shortID := generateShortID()
        urlStore[shortID] = req.URL

        resp := ShortenResponse{ShortURL: fmt.Sprintf("http://localhost:8080/resolve/%s",
shortID)}
        json.NewEncoder(w).Encode(resp)
}
```

Infine, creiamo l'handler per risolvere gli URL accorciati. Questo endpoint reindirizzerà l'utente all'URL originale.

```
func ResolveHandler(w http.ResponseWriter, r *http.Request) {
        shortID := r.URL.Path[len("/resolve/"):]

        originalURL, exists := urlStore[shortID]
        if !exists {
```

```
                http.NotFound(w, r)
                return
        }

        http.Redirect(w, r, originalURL, http.StatusFound)
}
```

Conclusione

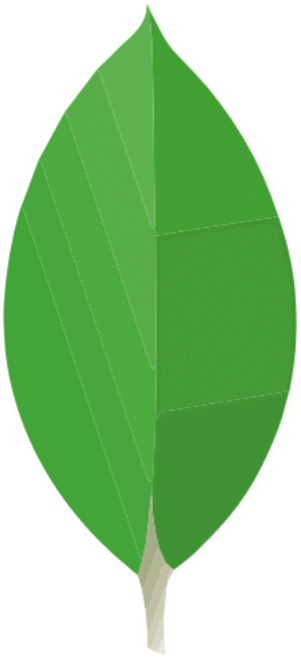
Abbiamo implementato un semplice URL shortener utilizzando Go. Il nostro sistema è molto basico e potrebbe essere esteso con funzionalità come la persistenza dei dati su un database, l'aggiunta di una scadenza per gli URL accorciati, e l'implementazione di un sistema di rate limiting. Tuttavia, questa guida fornisce una base solida per comprendere i concetti fondamentali e creare una versione personalizzata e più robusta di un URL shortener.

Applicazioni Correlate

-

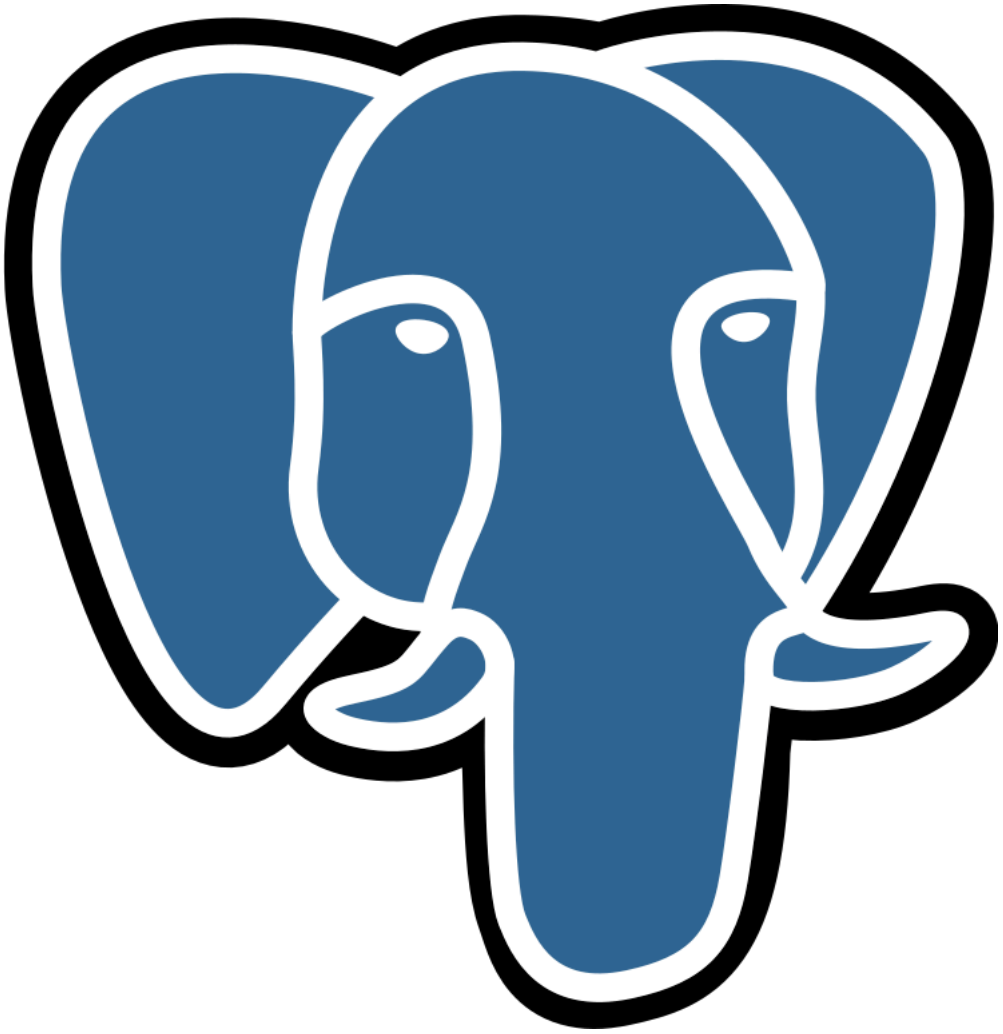
Go Placeholder Image

Applicazione in Go per la creazione di immagini segnaposto.
DockerDocker ComposeGoJavaScript



Go MongoDB App

Applicazione basata su MongoDB ed implementata in Go con il driver ufficiale.
DockerDocker ComposeGoMongoDB



Go PostgreSQL App

Applicazione basata su PostgreSQL e sviluppata in Go.
DockerDocker ComposeGoPostgreSQL