GABRIELE ROMANATO

Creare un'app per le richieste WHOIS dei domini .it con Angular e Go

In questo articolo vedremo come creare una web application per effettuare query WHOIS al Registro dei nomi a dominio .it utilizzando Angular come framework frontend e Go come soluzione per implementare le API del backend.

Anatomia di una richiesta WHOIS

1. Si effettua una connessione TCP al server WHOIS del registro italiano usando `whois.nic.it` come host e `43` come numero di porta remota.
2. Si invia il nome del dominio .it al server remoto seguito da un'interruzione di riga (caratteri `\r\n`).
3. Si riceve la risposta grezza, con i blocchi delle informazioni separati da una interruzione di riga (carattere `\n`).

Creare le API con Go

Utilizzeremo il framework Gin per creare le API REST in Go. Prima di implementare la route di riferimento, tuttavia, dobbiamo separare la logica della richiesta WHOIS dal routing definendo il package `utils` con le funzioni necessarie.

```
package utils

// utils/utils.go

import (
    "net"
    "regexp"
)
```

```
type WhoisResponse struct {
    Result string `json:"result"`
}

func ValidateDomainExtension(domain string) bool {
    m := regexp.MustCompile(`(?i)\.it$`)
    return m.MatchString(domain)
}

func WhoisRequest(domain, server string) WhoisResponse {
    conn, err := net.Dial("tcp", server+":43")
    if err != nil {
        return WhoisResponse{Result: "Error"}
    }
    defer conn.Close()
    conn.Write([]byte(domain + "\r\n"))
    buf := make([]byte, 1024)
    n, _ := conn.Read(buf)
    return WhoisResponse{Result: string(buf[:n])}
}
```

La funzione principale effettua la richiesta WHOIS restituendo la struct con il risultato che verrà poi restituita come JSON nella route che andremo a definire nel modulo `routes`.

```
package routes

// routes/routes.go

import (
    "tuo-namespace/tuo-nome-app/utils"

    "github.com/gin-gonic/gin"
)

const (
    whoisServer = "whois.nic.it"
)

func HandleWhoisRequest(c *gin.Context) {
    domain := c.Param("domain")
```

```
    if !utils.ValidateDomainExtension(domain) {
        c.JSON(400, gin.H{"error": "Invalid domain
extension"})
        return
    }
    whoisResponse := utils.WhoisRequest(domain, whoisServer)
    c.JSON(200, whoisResponse)
}
```

La funzione che gestisce l'endpoint effettua la validazione del nome a dominio passato come parametro della route, che deve avere l'estensione .it, e quindi restituisce il risultato in formato JSON.

Nel file principale della nostra applicazione andremo ad abilitare CORS per le richieste alle API e inizializzeremo le route e l'app stessa.

```
package main

import (
    "tuo-namespace/tuo-nome-app/routes"
    "time"

    "github.com/gin-contrib/cors"
    "github.com/gin-gonic/gin"
)

const (
    appPort = ":4000"
)

func main() {

    r := gin.Default()

    config := cors.DefaultConfig()
    config.AllowAllOrigins = true
    config.AllowMethods = []string{"POST", "GET", "PUT",
"OPTIONS"}
    config.AllowHeaders = []string{"Origin", "Content-Type",
"Authorization", "Accept", "User-Agent", "Cache-Control",
"Pragma"}
```

```
    config.ExposeHeaders = []string{"Content-Length"}
    config.AllowCredentials = true
    config.MaxAge = 12 * time.Hour

    r.Use(cors.New(config))

    api := r.Group("/api")

    api.GET("/whois/:domain", routes.HandleWhoisRequest)

    r.Run(appPort)
}
```

Creare il frontend con Angular

Per cominciare, dobbiamo creare una variabile di environment contenente l'URL di riferimento delle nostre API in Go.

```
// src/environments/environment.development.ts

export const environment = {
  production: false,
  apiUrl: 'http://localhost:4000/api'
};
```

Ora dobbiamo definire un DTO per gestire il formato di risposta delle API, che può contenere una proprietà `result` o una proprietà `error`. Per evitare di dover effettuare un type check ogni volta, le definiamo entrambe:

```
// src/app/core/dto/whois-response.dto.ts

export interface WhoisResponseDto {
  result: string;
  error: string;
}
```

La risposta sarà in formato grezzo che andrà parsato dal service dedicato per ottenere la struttura finale che definiremo nel model dedicato.

```
// src/app/core/models/whois-response.model.ts

export interface WhoisResponseModel {
    domain: string;
    status: string;
    created: string;
    updated: string;
    expires: string;
    registrar: {
      organization: string;
      name: string;
      web: string;
    }
}
```

Ora possiamo definire il service principale:

```
// src/app/core/services/whois.service.ts

mport { Injectable } from '@angular/core';
import { HttpClient } from "@angular/common/http";
import { Observable } from "rxjs";
import { WhoisResponseDto } from "../dto/whois-response.dto";
import { environment } from
"../../../environments/environment";
import { WhoisResponseModel } from "../models/whois-
response.model";

@Injectable({
  providedIn: 'root'
})
export class WhoisService {

  private readonly apiUrl = environment.apiUrl;
  constructor(private http : HttpClient) { }
```

```
  public parseWhoisData(whoisResult: string):
WhoisResponseModel {
      const lines = whoisResult.split('\n');
      const response : WhoisResponseModel = {
        domain: '',
        status: '',
        created: '',
        updated: '',
        expires: '',
        registrar: {
          organization: '',
          name: '',
          web: ''
        }
      };
      for(let i = 0; i < lines.length; i++) {
        let line = lines[i];
        if (line.includes('Domain:')) {
          response.domain = line.split(':')[1].trim();
        }
        if (line.includes('Status:')) {
          response.status = line.split(':')[1].trim();
        }
        if (line.includes('Created:')) {
          response.created =
line.replace(/\d{2}:\d{2}:\d{2}/g, '').split(':')[1].trim();
        }
        if (line.includes('Last Update:')) {
          response.updated =
line.replace(/\d{2}:\d{2}:\d{2}/g, '').split(':')[1].trim();
        }
        if (line.includes('Expire Date:')) {
          response.expires = line.split(':')[1].trim();
        }
        if
(/Registrant|Admin|Contacts|Registrar/gi.test(line)) {
          continue;
        }
        if(line.includes('Organization:')) {
          response.registrar.name = line.split(':')
[1].trim();
        }
        if(line.includes('Name:')) {
```

```
          response.registrar.organization = line.split(':')
[1].trim();
        }
        if(line.includes('Web:')) {
          response.registrar.web =
line.substring(line.indexOf(':') + 1).trim();
        }
      }
      return response;
  }

  public getWhoisData(domain: string):
Observable<WhoisResponseDto> {
    return this.http.get<WhoisResponseDto>
(`${this.apiUrl}/whois/${domain}`);
  }
}
```

Come si può notare, il metodo più laborioso è quello che effettua il parsing del risultato testuale ottenuto dalla richiesta WHOIS. Per farlo occorre procedere riga per riga individuando le sezioni che ci interessano e mappandole nel model di riferimento.

Vogliamo anche che l'utente possa avere una cronologia delle sue richieste. Per farlo utilizzeremo il web storage salvando una stringa JSON con le informazioni sui vari domini. Definiamo quindi un model per questo tipo di dato:

```
// src/app/core/models/whois-storage.model.ts

import {WhoisResponseModel} from "./whois-response.model";

export interface WhoisStorageModel {
  domains: WhoisResponseModel[];
}
```

Quindi definiamo il service di riferimento per la gestione del web storage:

```
// src/app/core/services/storage.service.ts

import { Injectable } from '@angular/core';
import { WhoisStorageModel } from "../models/whois-
storage.model";
import { WhoisResponseModel } from "../models/whois-
response.model";

@Injectable({
  providedIn: 'root'
})
export class StorageService {
  constructor() { }

  addDomainToStorage(domain: string, whoisResponse:
WhoisResponseModel) {
    if(this.isDomainInStorage(domain)) {
      return;
    }
    const domains = this.getDomainsFromStorage();
    domains.domains.push(whoisResponse);
    localStorage.setItem('domains', JSON.stringify(domains));
  }

  isDomainInStorage(domain: string): boolean {
    const domains = this.getDomainsFromStorage();
    return domains.domains.some(d => d.domain === domain);
  }

  removeDomainFromStorage(domain: string) {
    if(!this.isDomainInStorage(domain)) {
      return;
    }
    const domains = this.getDomainsFromStorage();
    domains.domains = domains.domains.filter(d => d.domain
!== domain);
    localStorage.setItem('domains', JSON.stringify(domains));
  }

  getDomainsFromStorage(): WhoisStorageModel {
    const domains = localStorage.getItem('domains');
    if (domains) {
```

```
      return JSON.parse(domains);
    }
    return { domains: [] };
  }

  getDomainFromStorage(domain: string): WhoisResponseModel {
    const domains = this.getDomainsFromStorage();
    if(domains.domains.length > 0) {
      return domains.domains.find(d => d.domain === domain)
|| {} as WhoisResponseModel;
    }
    return {} as WhoisResponseModel;
  }

  updateDomainInStorage(domain: WhoisResponseModel) {
    const domains = this.getDomainsFromStorage();
    domains.domains = domains.domains.map(d => {
      if(d.domain === domain.domain) {
        return domain;
      }
      return d;
    });
    localStorage.setItem('domains', JSON.stringify(domains));
  }
}
```

A livello di route, avremo essenzialmente due sezioni:

1. La home page, con il form di ricerca.
2. La pagina del listato della cronologia dei domini.

Poichè una richiesta può comportare un'attesa da parte dell'utente, definiamo un componente loader da mostrare durante l'attesa.

```
// src/app/shared/components/loader/loader.module.ts

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { LoaderComponent } from './loader.component';
```

```
@NgModule({
  declarations: [
    LoaderComponent
  ],
  imports: [
    CommonModule
  ],
  exports: [
    LoaderComponent
  ]
})
export class LoaderModule { }
```

```
// src/app/shared/components/loader/loader.component.ts

import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-loader',
  templateUrl: './loader.component.html',
  styleUrls: ['./loader.component.css']
})
export class LoaderComponent {
    @Input() visible = false;
}
```

```
<!-- src/app/shared/components/loader/loader.component.html--
>

<div class="app-loader" [class.visible]="visible">
    <div class="loader"></div>
</div>
```

```
/* src/app/shared/components/loader/loader.component.css */

.app-loader {
  position: fixed;
  top: 0;
```

```
  left: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.6);
  display: none;
  align-items: center;
  justify-content: center;
}

.app-loader.visible {
  display: flex;
}

.loader {
  border: 5px solid #fff;
  border-top: 5px solid #f36668;
  border-radius: 50%;
  width: 50px;
  height: 50px;
  animation: spin 2s linear infinite;
}

@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}
```

Ora possiamo definire il componente della home page:

```
// src/app/features/home/home.component.ts

import {Component} from '@angular/core';
import {WhoisService} from
'../../core/services/whois.service';
import {StorageService} from
"../../core/services/storage.service";
import {WhoisResponseModel} from '../../core/models/whois-
```

```
response.model';
import {WhoisResponseDto} from '../../core/dto/whois-
response.dto';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css'],
  providers: [WhoisService, StorageService]
})
export class HomeComponent {

  domain = '';
  whoisResponse: WhoisResponseModel = {} as
WhoisResponseModel;
  errorMessage = '';
  showLoader = false;

  constructor(private whoisService: WhoisService, private
storageService: StorageService) {
  }

  handleWhoisSearch() {
    this.errorMessage = '';
    this.whoisResponse = {} as WhoisResponseModel;
    this.showLoader = true;

    if (this.storageService.isDomainInStorage(this.domain)) {
      this.whoisResponse =
this.storageService.getDomainFromStorage(this.domain);
      this.showLoader = false;
      return;
    }
    this.whoisService.getWhoisData(this.domain).subscribe({
      next: (response: WhoisResponseDto) => {
        this.whoisResponse =
this.whoisService.parseWhoisData(response.result);
        this.showLoader = false;
        this.storageService.addDomainToStorage(this.domain,
this.whoisResponse);
        window.location.href = '/domains';
      },
      error: (error) => {
```

```
        this.errorMessage = error.error.error;
        this.showLoader = false;
      }
    });
  }
}
```

Se il dominio è già presente nella cronologia, restituiamo immediatamente il risultato. Il relativo template HTML potrebbe essere il seguente:

```
<!-- src/app/features/home/home.component.html --><

<form (ngSubmit)="handleWhoisSearch()">
    <div class="form-group">
        <input [(ngModel)]="domain" type="text" id="domain"
name="domain" placeholder="Enter a domain name">
        <button type="submit" class="btn btn-
primary">Whois</button>
    </div>
    <div *ngIf="errorMessage" class="alert alert-danger">
{{errorMessage}}</div>
    <div class="app-whois-result"
*ngIf="whoisResponse.domain">
        <div class="app-whois-result__item">
            <div class="app-whois-result__label">Domain</div>
            <div class="app-whois-result__value">
{{whoisResponse.domain}}</div>
        </div>
        <div class="app-whois-result__item">
            <div class="app-whois-result__label">Status</div>
            <div class="app-whois-result__value">
{{whoisResponse.status}}</div>
        </div>
        <div class="app-whois-result__item">
            <div class="app-whois-
result__label">Registrar</div>
            <div class="app-whois-result__value">
{{whoisResponse.registrar.name}}</div>
        </div>
        <div class="app-whois-result__item">
```

```
            <div class="app-whois-
result__label">Created</div>
            <div class="app-whois-result__value">
{{whoisResponse.created}}</div>
        </div>
        <div class="app-whois-result__item">
            <div class="app-whois-
result__label">Updated</div>
            <div class="app-whois-result__value">
{{whoisResponse.updated}}</div>
        </div>
        <div class="app-whois-result__item">
            <div class="app-whois-
result__label">Expires</div>
            <div class="app-whois-result__value">
{{whoisResponse.expires}}</div>
        </div>
    </div>
</form>
<app-loader [visible]="showLoader"></app-loader>
```

La pagina dei domini permetterà all'utente di visualizzare i domini salvati, rimuoverli dalla cronologia o aggiornarli effettuando una nuova richiesta WHOIS. Il componente di riferimento è il seguente:

```
// src/app/features/domains/domains.component.ts

import { Component } from '@angular/core';
import { StorageService } from
"../../core/services/storage.service";
import { WhoisStorageModel } from "../../core/models/whois-
storage.model";
import { WhoisService } from
"../../core/services/whois.service";
import { WhoisResponseModel } from "../../core/models/whois-
response.model";
import { WhoisResponseDto } from "../../core/dto/whois-
response.dto";

@Component({
```

```
  selector: 'app-domains',
  templateUrl: './domains.component.html',
  styleUrls: ['./domains.component.css'],
  providers: [StorageService, WhoisService]
})
export class DomainsComponent {
    data: WhoisStorageModel = {} as WhoisStorageModel;
    loaderVisible = false;

    constructor(private storageService: StorageService,
private whoisService: WhoisService) {
    }

    removeDomain(domain: string) {
      this.storageService.removeDomainFromStorage(domain);
      this.data =
this.storageService.getDomainsFromStorage();
    }

    updateDomain(domain: WhoisResponseModel) {
      const domainName = domain.domain;
      this.loaderVisible = true;
      this.whoisService.getWhoisData(domainName).subscribe({
        next: (response: WhoisResponseDto) => {
          const whoisResponse =
this.whoisService.parseWhoisData(response.result);

this.storageService.updateDomainInStorage(whoisResponse);
          this.data =
this.storageService.getDomainsFromStorage();
          this.loaderVisible = false;
        },
        error: (error) => {
          console.log(error);
          this.loaderVisible = false;
        }
      });
    }

    ngOnInit() {
      this.data =
this.storageService.getDomainsFromStorage();
```

```
    }
}
```

```
// src/app/features/domains/domains.module.ts

import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DomainsComponent } from './domains.component';
import { LoaderModule } from
"../../shared/components/loader/loader.module";


@NgModule({
  declarations: [
    DomainsComponent
  ],
  imports: [
    CommonModule,
    LoaderModule
  ],
  exports: [
    DomainsComponent
  ]
})
export class DomainsModule { }
```

Il template HTML di riferimento potrebbe essere il seguente:

```
<!-- src/app/features/domains/domains.component.html -->

<section class="domains">
    <header>
        <h1>Domains</h1>
    </header>
    <p *ngIf="data.domains.length === 0">No domains found</p>
     <ul *ngIf="data.domains.length > 0">
          <li *ngFor="let domain of data.domains.reverse()">
            <div class="domain">
              <div class="name">{{ domain.domain }}</div>
```

```
              <div><strong>Status:</strong> <span>{{
domain.status }}</span></div>
              <div>
                <strong>Expires:</strong> <span>{{
domain.expires }}</span>
              </div>
              <div>
                <strong>Created:</strong> <span>{{
domain.created }}</span>
              </div>
              <div>
                <strong>Updated:</strong> <span>{{
domain.updated }}</span>
              </div>
              <div>
                <strong>Registrar:</strong> <span>{{
domain.registrar.name }}</span>
              </div>
              <div class="domain-actions">
                <button class="remove-domain"
(click)="removeDomain(domain.domain)">Remove</button>
                <button class="update-domain"
(click)="updateDomain(domain)">Update</button>
              </div>

            </div>
          </li>
      </ul>
</section>
<app-loader [visible]="loaderVisible"></app-loader>
```

Infine, abilitiamo le nostre route nel file dedicato della nostra applicazione:

```
// src/app/core/components/app/app-routing.module.ts

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from
'../../../features/home/home.component';
import { DomainsComponent } from
'../../../features/domains/domains.component';
```

```
const routes: Routes = [{
  path: '',
  component: HomeComponent,
  title: 'Whois NIC IT'
}, {
  path: 'domains',
  component: DomainsComponent,
  title: 'Domains - Whois NIC IT'
}];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Non dimentichiamo ovviamente di modificare il modulo principale della nostra app:

```
// src/app/core/components/app/app.module.ts

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from "@angular/common/http";

import { AppRoutingModule } from './app-routing.module';
import { HomeModule } from
"../../../features/home/home.module";
import { DomainsModule } from
"../../../features/domains/domains.module";
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
```

```
    HttpClientModule,
    HomeModule,
    DomainsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Conclusioni

Per quanto a prima vista il compito potrebbe apparire arduo, in realtà una volta compresa l'anatomia di una richiesta WHOIS e l'output prodotto, sviluppare un'applicazione che ne faccia uso è un compito relativamente lineare con Angular e Go.