Gestione del CORS in Go con Gin: un middleware personalizzato

Quando si sviluppano API che devono essere consumate da frontend ospitati su domini differenti, è essenziale configurare correttamente il CORS (Cross-Origin Resource Sharing). In questo articolo vediamo come realizzare un middleware in Go utilizzando il framework `gin` per abilitare il CORS in modo selettivo.

Il Codice del Middleware

Il seguente snippet implementa un middleware chiamato `CorsMiddleware`:

```
func CorsMiddleware() gin.HandlerFunc {
        originsString := "https://example.com,https://api.example.com"
        var allowedOrigins []string
        if originsString != "" {
                allowedOrigins = strings.Split(originsString, ",")
        }

        return func(c *gin.Context) {
                isOriginAllowed := func(origin string, allowedOrigins []string) bool {
                        for _, allowedOrigin := range allowedOrigins {
                                if origin == allowedOrigin {
                                        return true
                                }
                        }
                        return false
                }

                origin := c.Request.Header.Get("Origin")

                if isOriginAllowed(origin, allowedOrigins) {
                        c.Writer.Header().Set("Access-Control-Allow-Origin", origin)
                        c.Writer.Header().Set("Access-Control-Allow-Credentials", "true")
                        c.Writer.Header().Set("Access-Control-Allow-Headers", "Content-Type,
Content-Length, Accept-Encoding, X-CSRF-Token, Authorization, accept, origin, Cache-Control, X-
Requested-With")
                        c.Writer.Header().Set("Access-Control-Allow-Methods", "POST, OPTIONS,
GET, PUT, DELETE, PATCH")
                }

                if c.Request.Method == "OPTIONS" {
                        c.AbortWithStatus(204)
                        return
                }

                c.Next()
        }
}
```

Come Funziona

Il middleware parte da una stringa contenente i domini autorizzati separati da virgola. Questa stringa viene trasformata in una slice di stringhe con `strings.Split`.

All'interno della funzione middleware, per ogni richiesta in arrivo, viene controllato l'header `Origin`. Se l'origine è presente tra quelle consentite, vengono impostati gli header necessari per abilitare il CORS:

- `Access-Control-Allow-Origin`: imposta l'origine autorizzata.
- `Access-Control-Allow-Credentials`: permette l'invio di cookie o credenziali.
- `Access-Control-Allow-Headers`: specifica gli header permessi.
- `Access-Control-Allow-Methods`: indica i metodi HTTP accettati.

Infine, se la richiesta è di tipo `OPTIONS` (preflight request), viene interrotta con uno status code `204`, come previsto dallo standard CORS.

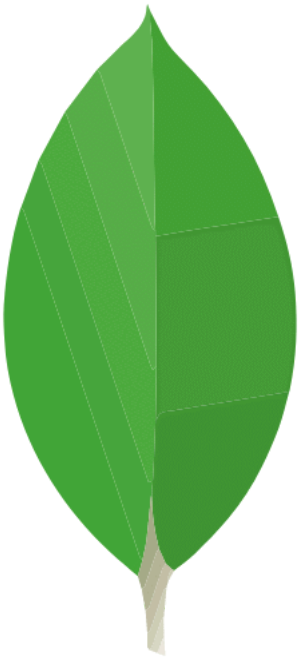
Conclusione

Questo middleware offre un controllo fine sull'accesso cross-origin, utile in contesti di produzione dove è necessario specificare esattamente quali frontend possono accedere alle API. È un esempio semplice ma potente di come Gin possa essere esteso per gestire scenari reali in modo elegante.

Applicazioni Correlate

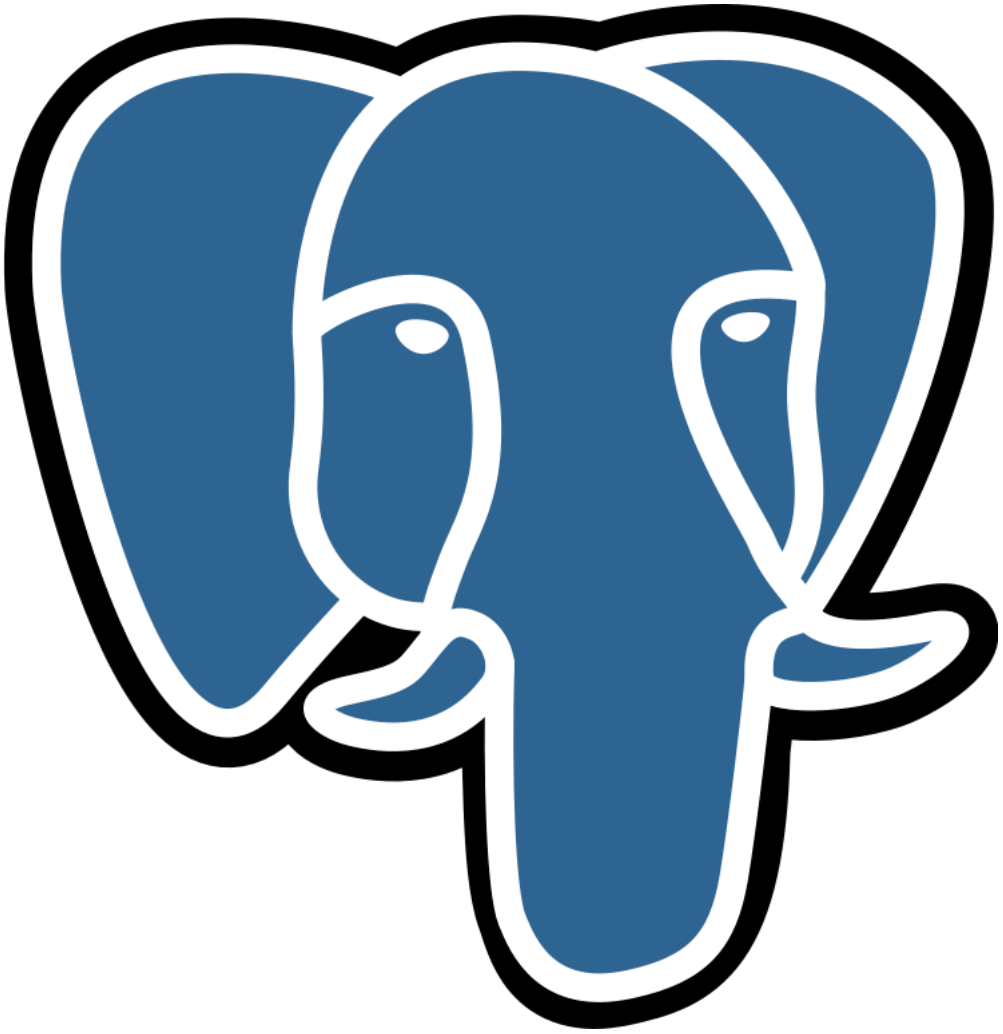
- **Go Placeholder Image**

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