Come implementare CORS da zero in Java con Spring

Cross-Origin Resource Sharing (CORS) è un meccanismo che consente alle risorse di un'applicazione web di essere richieste da un dominio differente rispetto a quello su cui l'applicazione è ospitata. In un'applicazione Spring Boot, puoi implementare CORS manualmente per avere pieno controllo su quali origini, metodi e intestazioni sono consentiti.

1. CORS a livello globale tramite un filtro personalizzato

Il primo approccio consiste nel creare un filtro che intercetta tutte le richieste e gestisce le intestazioni CORS:

```
import jakarta.servlet.Filter;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.ServletRequest;
import jakarta.servlet.ServletResponse;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.stereotype.Component;

import java.io.IOException;

@Component
public class CustomCorsFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request,
ServletResponse response, FilterChain chain)
            throws IOException, ServletException {
```

```
        HttpServletResponse res = (HttpServletResponse)
response;
        HttpServletRequest req = (HttpServletRequest)
request;

        res.setHeader("Access-Control-Allow-Origin",
"https://example.com");
        res.setHeader("Access-Control-Allow-Methods", "GET,
POST, PUT, DELETE, OPTIONS");
        res.setHeader("Access-Control-Allow-Headers",
"Content-Type, Authorization");
        res.setHeader("Access-Control-Allow-Credentials",
"true");

        // Gestione preflight
        if ("OPTIONS".equalsIgnoreCase(req.getMethod())) {
            res.setStatus(HttpServletResponse.SC_OK);
        } else {
            chain.doFilter(request, response);
        }
    }
}
```

2. CORS a livello di controller con `@CrossOrigin`

Un modo più semplice per abilitare CORS su specifici controller o metodi è usare l'annotazione `@CrossOrigin`:

```
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
public class ExampleController {

    @CrossOrigin(origins = "https://example.com",
allowCredentials = "true")
    @GetMapping("/dati")
```

```
    public String getDati() {
        return "Risposta dal server";
    }
}
```

3. Configurazione CORS globale con

`WebMvcConfigurer`

Per una configurazione globale e centralizzata, puoi implementare l'interfaccia `WebMvcConfigurer`:

```
import org.springframework.context.annotation.Configuration;
import
org.springframework.web.servlet.config.annotation.CorsRegistr
y;
import
org.springframework.web.servlet.config.annotation.WebMvcConfi
gurer;

@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("https://example.com")
                .allowedMethods("GET", "POST", "PUT",
"DELETE", "OPTIONS")
                .allowedHeaders("Content-Type",
"Authorization")
                .allowCredentials(true);
    }
}
```

Conclusione

Implementare CORS da zero in un'applicazione Spring offre grande flessibilità. Scegli l'approccio più adatto al tuo caso d'uso: un filtro personalizzato per un controllo totale, `@CrossOrigin` per configurazioni rapide per endpoint specifici, o `WebMvcConfigurer` per un setup globale semplice e manutenibile.