

Configuración CORS - Sistema Tiendario

¿Qué es CORS?

CORS (Cross-Origin Resource Sharing) es un mecanismo de seguridad implementado por los navegadores web que controla cómo las páginas web pueden realizar solicitudes a dominios diferentes al que sirve la página web.

Problema que resuelve CORS

- **Same-Origin Policy:** Los navegadores bloquean requests entre diferentes orígenes por seguridad
- **Orígenes diferentes:** Diferentes protocolos, dominios o puertos
- **Necesidad:** Permitir que frontends accedan a nuestra API REST

Configuración CORS Implementada

Ubicación de la Configuración

Archivo: SecurityConfig.java **Método:** corsFilter()

Código de Configuración

```
java

@Bean
public CorsFilter corsFilter() {
    CorsConfiguration config = new CorsConfiguration();

    config.setAllowedOrigins(List.of("http://localhost:3000"));

    config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
    config.setAllowedHeaders(List.of("*"));

    config.setAllowCredentials(true);

    UrlBasedCorsConfigurationSource source = new
    UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", source);
    return new CorsFilter(source);
}
```

Configuración Detallada

1. Orígenes Permitidos (Allowed Origins)

```
java
config.setAllowedOrigins(List.of("http://localhost:3000"));
```

Configuración:

- ☒ http://localhost:3000 - Frontend en desarrollo

Propósito:

- Permite que aplicaciones React/Angular/Vue accedan a la API
- Bloquea requests desde otros dominios no autorizados
- Específico para desarrollo local

2. Métodos HTTP Permitidos (Allowed Methods)

```
java
config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
```

Métodos habilitados:

- ☒ **GET** - Consultas (listar usuarios, productos, etc.)
- ☒ **POST** - Creación (crear usuario, producto, etc.)
- ☒ **PUT** - Actualización completa
- ☒ **DELETE** - Eliminación
- ☒ **OPTIONS** - Preflight requests (automático del navegador)

Cobertura: Todos los métodos CRUD necesarios para la API REST

3. Headers Permitidos (Allowed Headers)

```
java
config.setAllowedHeaders(List.of("*"));
```

Headers importantes:

- ☒ Authorization - Para tokens JWT
- ☒ Content-Type - Para JSON requests
- ☒ Accept - Para especificar formato de respuesta
- ☒ Todos los headers (*) - Máxima flexibilidad

4. Credenciales Habilitadas (Allow Credentials)

```
java
config.setAllowCredentials(true);
```

Función:

- ☒ Permite envío de cookies
- ☒ Permite headers de autorización (JWT)
- ☒ Mantiene sesiones entre requests
- ⚠ **Importante:** No se puede usar con `allowedOrigins("*")`

5. Configuración Global (URL Pattern)

```
java
source.registerCorsConfiguration("/**", config);
```

Alcance:

- Aplica a **todos** los endpoints (/**)
- Incluye `/api/usuarios`, `/api/productos`, `/login`, etc.
- Configuración uniforme en toda la aplicación

Escenarios de Uso CORS

Desarrollo Local

Frontend: `http://localhost:3000` (React/Angular)

Backend: `http://localhost:8080` (Spring Boot)

Problema: Diferentes puertos = Diferentes orígenes

Solución: CORS configurado para permitir `localhost:3000`

Validación Cruzada con Ngrok

Frontend: <http://localhost:3000>

Backend: `https://abc123.ngrok.io` (URL pública temporal)

Problema: Diferentes protocolos y dominios

Solución: CORS permite requests cross-origin

Producción (Ejemplo)

Frontend: <https://tiendario-frontend.com>

Backend: <https://api.tiendario.com>

Configuración necesaria:

```
java
config.setAllowedOrigins(List.of("https://tiendario-frontend.com"));
```

🔍 Preflight Requests

¿Qué son?

Requests automáticos que hace el navegador antes del request real para verificar permisos CORS.

Cuándo ocurren:

- Métodos diferentes a GET, POST simple
- Headers personalizados (como Authorization)
- Content-Type application/json

Flujo típico:

1. **Navegador:** Envía OPTIONS request (preflight)
2. **Servidor:** Responde con headers CORS permitidos
3. **Navegador:** Si OK, envía el request real
4. **Servidor:** Procesa y responde normalmente

Ejemplo de preflight:

```
http
OPTIONS /api/usuarios HTTP/1.1
Host: localhost:8080
Origin: http://localhost:3000
Access-Control-Request-Method: POST
Access-Control-Request-Headers: authorization,content-type
```

Respuesta del servidor:

```
http
HTTP/1.1 200 OK
Access-Control-Allow-Origin: http://localhost:3000
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
Access-Control-Allow-Headers: *
Access-Control-Allow-Credentials: true
```

❑ Pruebas CORS

Herramientas de Prueba

1. **Navegador:** Consola de desarrollador (F12)
2. **Postman:** Simula requests cross-origin
3. **Curl:** Tests desde línea de comandos
4. **Frontend real:** React/Angular/Vue

Comandos de Prueba

```
bash
# Preflight request manual

curl -X OPTIONS \
  -H "Origin: http://localhost:3000" \
  -H "Access-Control-Request-Method: POST" \
  -H "Access-Control-Request-Headers: authorization,content-type" \
  http://localhost:8080/api/usuarios


# Request real con CORS

curl -X GET \
  -H "Origin: http://localhost:3000" \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cGE6IjoiIn0=" \
  http://localhost:8080/api/usuarios
```

✖ Errores CORS Comunes

1. Origen no permitido

Access to XMLHttpRequest at 'http://localhost:8080/api/usuarios' from origin 'http://localhost:3001' has been blocked by CORS policy

Solución: Agregar http://localhost:3001 a allowedOrigins

2. Método no permitido

Method PUT is not allowed by Access-Control-Allow-Methods

Solución: Agregar "PUT" a allowedMethods

3. Header no permitido

Request header authorization is not allowed by Access-Control-Allow-Headers

Solución: Agregar "authorization" a allowedHeaders o usar ""

4. Credenciales con wildcard

Cannot use wildcard in Access-Control-Allow-Origin when credentials flag is true

Solución: Especificar origen exacto en lugar de ""

Configuración para Producción

Recomendaciones de Seguridad

```
java

// Producción - Orígenes específicos

config.setAllowedOrigins(List.of(
    "https://tiendario-frontend.com",
    "https://admin.tiendario.com"
));

// Headers específicos (más seguro que "**")

config.setAllowedHeaders(List.of(
    "Authorization",
    "Content-Type",
    "Accept",
    "X-Requested-With"
));

// Métodos mínimos necesarios

config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE"));
```

Variables de Entorno

```
java

// Mejor práctica - Configuración desde environment

@Value("${app.cors.allowed-origins}")
private List<String> allowedOrigins;





config.setAllowedOrigins(allowedOrigins);
```


Resumen de Configuración Actual





Configuración	Valor	Propósito
Allowed Origins	http://localhost:3000	Frontend desarrollo
Allowed Methods	GET, POST, PUT, DELETE, OPTIONS	CRUD completo
Allowed Headers	* (todos)	Máxima flexibilidad
Allow Credentials	true	JWT Authorization
URL Pattern	/**	Toda la aplicación

Beneficios de la Configuración





Para Desarrollo

-  Frontend puede consumir API sin problemas
-  Herramientas como Postman funcionan correctamente
-  Swagger UI accesible desde navegador
-  Testing cross-origin simplificado

Para Validación Cruzada

-  Otros equipos pueden acceder via ngrok
-  Colecciones Postman funcionan directamente
-  No hay bloqueos por políticas CORS
-  Interoperabilidad garantizada

Para Producción

-  Base sólida para configuración segura
-  Fácil adaptación a dominios reales
-  Control granular de acceso
-  Cumplimiento de estándares web