

Para habilitar **CORS** (Cross-Origin Resource Sharing) em uma API REST feita com **Spring Boot**, você pode fazer isso de algumas maneiras, dependendo do escopo que deseja aplicar (global ou específico por controller/método).

✓ 1. Habilitar CORS Globalmente (recomendado para API pública)

Crie uma classe de configuração:

java

CopiarEditar

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
```

@Configuration

```
public class CorsConfig {
```


@Bean

```
public WebMvcConfigurer corsConfigurer() {
    return new WebMvcConfigurer() {
```

@Override

```
public void addCorsMappings(CorsRegistry registry) {
    registry.addMapping("/**") // permite todos os endpoints
        .allowedOrigins("*") // permite todas as origens. Substitua por
domínios específicos se necessário
        .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS") //
métodos permitidos
        .allowedHeaders("*"); // cabeçalhos permitidos
    }
}
```

```
};  
}  
}
```

 **Atenção:** Use `allowedOrigins("*")` apenas para testes ou APIs públicas. Em produção, defina os domínios permitidos, ex:
`allowedOrigins("https://meudominio.com").`

2. Habilitar CORS em um Controller específico

java

CopiarEditar

```
import org.springframework.web.bind.annotation.*;  
  
@RestController  
@RequestMapping("/api/exemplo")  
@CrossOrigin(origins = "*", methods = {RequestMethod.GET,  
RequestMethod.POST})  
public class ExemploController {  
  
    @GetMapping  
    public String exemplo() {  
        return "Hello CORS";  
    }  
}
```

3. Habilitar CORS em um método específico

java

CopiarEditar

```
@GetMapping("/dados")  
@CrossOrigin(origins = "https://dominio.com")
```

```
public ResponseEntity<String> dados() {  
    return ResponseEntity.ok("Dados com CORS");  
}
```

Se estiver usando **Spring Security**, há uma configuração extra a fazer. Posso te mostrar como ajustar também a segurança para permitir CORS, se for o caso.

Quer incluir isso também?

Você disse:

como aplicar segurança em api rest com springboot

O ChatGPT disse:

Para aplicar **segurança em uma API REST com Spring Boot**, o caminho mais comum e robusto é usar **Spring Security**. Você pode configurar autenticação, autorização, e proteger endpoints com base em roles, tokens JWT, ou até autenticação básica.

Abaixo está um guia **completo e prático** para proteger uma API usando **Spring Boot 3+**, com foco em:

Exemplo básico com Spring Security + JWT

1. Adicione as dependências no pom.xml (se for Maven)

xml

CopiarEditar

```
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-security</artifactId>  
</dependency>  
  
<dependency>  
    <groupId>io.jsonwebtoken</groupId>  
    <artifactId>jjwt-api</artifactId>  
    <version>0.11.5</version>
```

```
</dependency>

<dependency>

    <groupId>io.jsonwebtoken</groupId>

    <artifactId>jjwt-impl</artifactId>

    <version>0.11.5</version>

    <scope>runtime</scope>

</dependency>

<dependency>

    <groupId>io.jsonwebtoken</groupId>

    <artifactId>jjwt-jackson</artifactId>

    <version>0.11.5</version>

    <scope>runtime</scope>

</dependency>
```

2. Crie uma configuração de segurança

java

CopiarEditar

```
import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import
org.springframework.security.config.annotation.authentication.configuration.Auth
enticationConfiguration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.SecurityFilterChain;
```

@Configuration

```
public class SecurityConfig {
```

@Bean

```
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws  
Exception {
```

```
    http
```

```
        .csrf(csrf -> csrf.disable())
```

```
        .sessionManagement(sess ->  
sess.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
```

```
        .authorizeHttpRequests(auth -> auth
```

```
            .requestMatchers("/api/auth/**").permitAll() // rota pública
```

```
            .anyRequest().authenticated()
```

```
        );
```

```
    return http.build();
```

```
}
```

@Bean

```
public PasswordEncoder passwordEncoder() {
```

```
    return new BCryptPasswordEncoder();
```

```
}
```

@Bean

```
public AuthenticationManager
```

```
authenticationManager(AuthenticationConfiguration config) throws Exception {
```

```
    return config.getAuthenticationManager();
```

```
}
```

```
}
```

3. Crie um endpoint de autenticação (login)

java

CopiarEditar

@RestController

@RequestMapping("/api/auth")

public class AuthController {

@PostMapping("/login")

public ResponseEntity<?> login(@RequestBody LoginRequest request) {

// Validar usuário/senha, gerar e retornar um token JWT

return ResponseEntity.ok("TOKEN_JWT_AQUI");

}

}

4. Crie filtros e utilitários para JWT

Aqui você implementa um `JwtAuthenticationFilter` que verifica o token em cada requisição e autentica o usuário.

Se quiser, posso te mostrar como criar esse filtro e o utilitário JWT. Deseja isso também?

5. Protegendo endpoints

Agora, qualquer rota `**fora do /api/auth/**` exige token JWT válido.



Outras opções de segurança

- Autenticação via **Basic Auth** (simples, mas insegura se não usar HTTPS).
- Autenticação com **OAuth2/OpenID Connect** (Google, Facebook, Keycloak).

- Autenticação com **Session + FormLogin** (mais comum em apps web com UI).

Quer um exemplo mais completo com JWT funcionando do login ao uso do token? Posso gerar pra você um projeto modelo.

Você disse:

pode me dar exemplo usando basic auth

O ChatGPT disse:

Claro! Aqui está um exemplo completo de como proteger uma **API REST com Spring Boot usando Basic Auth**.

✅ 1. Dependência no pom.xml

Se estiver usando **Spring Boot com Maven**, adicione:

xml

CopiarEditar

```
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
```

```
    <artifactId>spring-boot-starter-security</artifactId>
```

```
</dependency>
```

✅ 2. Configurando um usuário em memória (exemplo simples)

Crie uma classe de configuração:

java

CopiarEditar

```
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
```

```
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
```

```
import org.springframework.security.core.userdetails.User;
```

```
import org.springframework.security.core.userdetails.UserDetailsService;
```

```
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
```

@Configuration

```
public class SecurityConfig {
```

@Bean

```
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
    Exception {
```

```
        http
```

```
            .csrf(csrf -> csrf.disable())
```

```
            .authorizeHttpRequests(auth -> auth
```

```
                .requestMatchers("/publico").permitAll()
```

```
                .anyRequest().authenticated()
```

```
            )
```

```
            .httpBasic(); // Habilita autenticação Basic Auth
```

```
        return http.build();
```

```
    }
```

@Bean

```
    public UserDetailsService userDetailsService(PasswordEncoder encoder) {
```

```
        var user = User.withUsername("admin")
```

```
            .password(encoder.encode("1234"))
```

```
            .roles("USER")
```

```
            .build();
```



```
        return new InMemoryUserDetailsManager(user);
    }
}
```

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
}
}
```

3. Criando um controller com rotas públicas e protegidas

java

CopiarEditar

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

```
@RestController
```

```
public class HelloController {
```

```
    @GetMapping("/publico")
    public String publico() {
        return "Rota pública: não precisa de login.";
    }
}
```

```
    @GetMapping("/protegido")
    public String protegido() {
        return "Rota protegida: você está autenticado!";
    }
}
```

}

✅ 4. Testando com Basic Auth

Use um cliente como **Postman**, **curl**, ou via browser.

Exemplo com curl:

bash

CopiarEditar

```
curl -u admin:1234 http://localhost:8080/protegido
```

🔒 Resultado:

- /publico → Acesso livre.
- /protegido → Requer autenticação com o usuário admin e senha 1234.