CORS

IN SPRING BOOT

ROHAN THAPA
thaparohan2019@gmail.com

What is CORS?

CORS is a **security feature** implemented by web browsers to prevent web applications from making requests to a **domain different** from the one that served the web page.

This is essential because it helps **protect users** from potentially malicious actions by restricting cross-origin HTTP requests.

Why CORS Matters

When you're building a **RESTful API** with **Spring Boot**, your frontend (which might be served from a different **domain** or **port**) often needs to communicate with your backend.

CORS must be properly configured to allow these **cross-origin** requests.

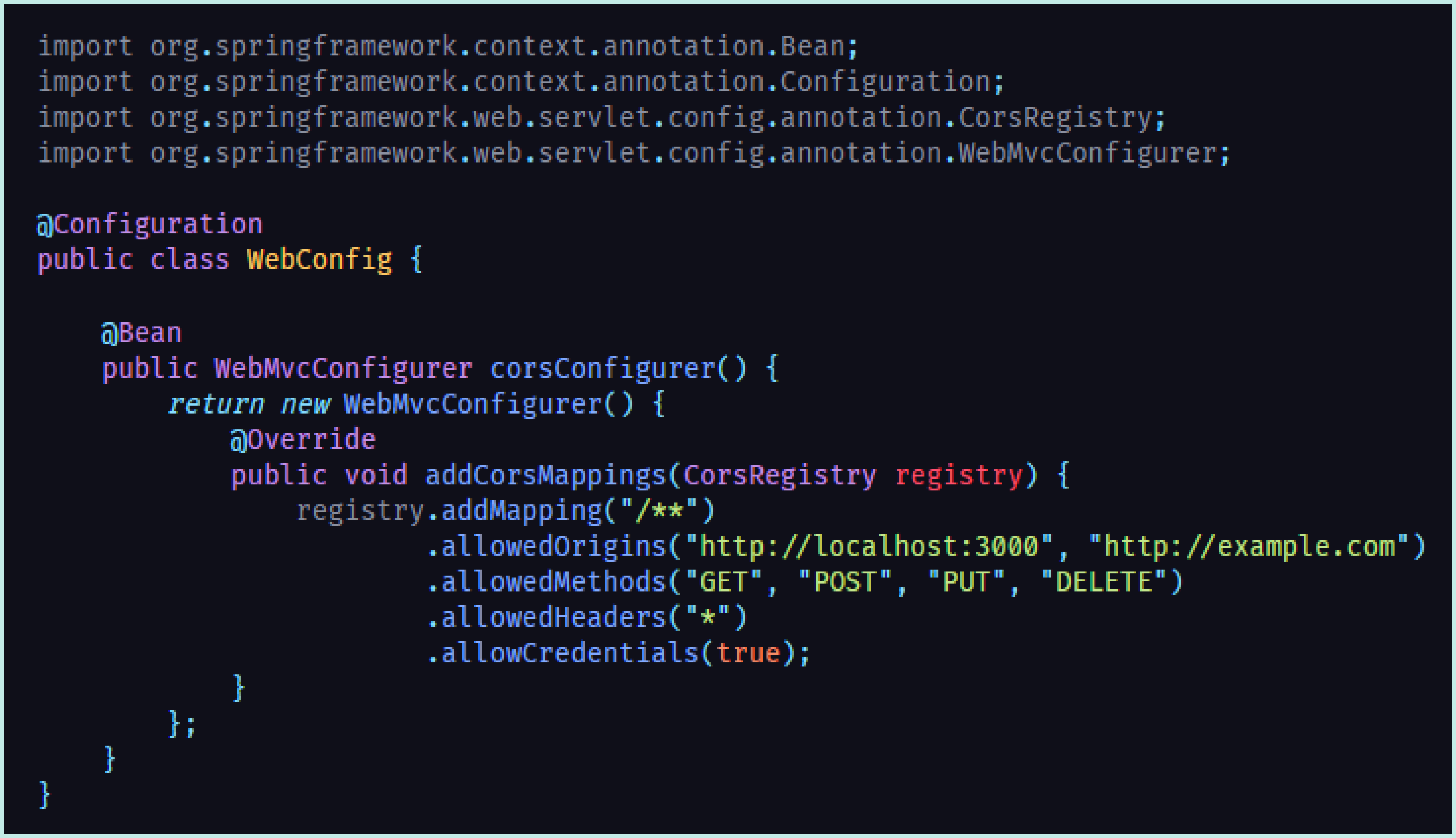
How Spring Boot Handles CORS

Spring Boot simplifies the configuration of CORS using **annotations** or **global configurations**.

1. Global CORS Configuration
2. Controller-level CORS Configuration

Global CORS Configuration

In Spring Boot , you can define global **CORS configurations** that apply to **all endpoints** in your application. This is particularly useful if you have a **consistent policy** across your API.



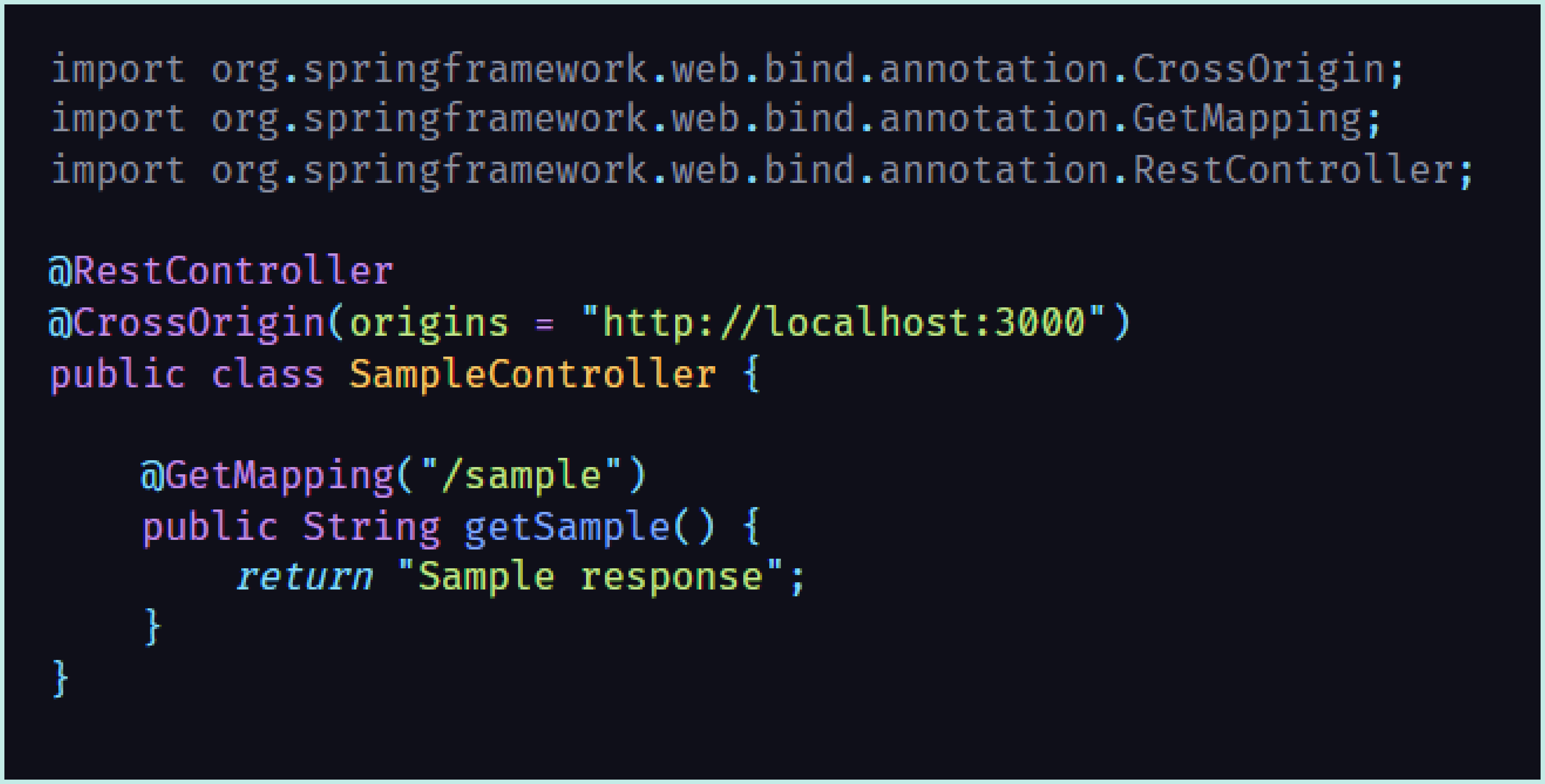
```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig {

    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedOrigins("http://localhost:3000", "http://example.com")
                        .allowedMethods("GET", "POST", "PUT", "DELETE")
                        .allowedHeaders("*")
                        .allowCredentials(true);
            }
        };
    }
}
```

Explanation

- **allowedOrigins:** Specifies the domains allowed to access your resources.
- **allowedMethods:** Specifies which **HTTP** methods (**GET, POST**, etc.) are allowed.
- **allowedHeaders:** Specifies which HTTP headers can be used in the actual request.
- **allowCredentials:** Allows the browser to include credentials **(like cookies)** in the request.

Controller-Level CORS Config

If you only need to apply **CORS** to **specific controllers** or **endpoints**, you can use the **@CrossOrigin** annotation directly in your controller classes or methods.



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

@RestController
@CrossOrigin(origins = "http://localhost:3000")
public class SampleController {

    @GetMapping("/sample")
    public String getSample() {
        return "Sample response";
    }
}
```

This allows only the specified origin (**http://localhost:3000** in this case) to access this endpoint.

Use Case

Imagine you're developing a **single-page application (SPA)** using React (running **on http://localhost:3000)** that interacts with a Spring Boot backend (running on **http://localhost:8080**).

Without proper **CORS** configuration, your **browser** will **block requests** from the React app to the Spring Boot API.

By setting up CORS, you ensure that these requests are **allowed**, enabling seamless interaction between the frontend and backend.

Conclusion

Understanding and configuring CORS is crucial for developing secure and functional web applications. Spring Boot provides powerful and flexible tools to manage CORS, whether at a global level or for specific endpoints.

Thank You

ROHAN THAPA
thaparohan2019@gmail.com