CORS and Versioning : Cross origin resource sharing

What does it mean ?

It is browser security feature that restricts cross-origin HTTP Request .

What qualifies fro Cross Origin HTTP Request ?

1.A different domain(eg from cat.com to dog.com) 2.A different subdomain(eg cat.com to adopt.cat.com) 3. Adifferent port ( eg from cat.com to cat.com:10700) 4. A different portocal (eg. From <https://cat.com> to <http://cat.com>) .

Cross-Origin Resource Sharing (CORS) is a security concept that allows restricting the resources implemented in web browsers. It prevents the JavaScript code producing or consuming the requests against different origin.

For example, your web application is running on 8080 port and by using JavaScript you are trying to consuming RESTful web services from 9090 port. Under such situations, you will face the Cross-Origin Resource Sharing security issue on your web browsers.

Two requirements are needed to handle this issue −

* RESTful web services should support the Cross-Origin Resource Sharing.
* RESTful web service application should allow accessing the API(s) from the 8080 port.

In this chapter, we are going to learn in detail about How to Enable Cross-Origin Requests for a RESTful Web Service application.

Enable CORS in Controller Method

We need to set the origins for RESTful web service by using **@CrossOrigin** annotation for the controller method. This @CrossOrigin annotation supports specific REST API, and not for the entire application.

@RequestMapping(value = "/products")

@CrossOrigin(origins = "http://localhost:8080")

public ResponseEntity<Object> getProduct() {

return null;

}

Global CORS Configuration : This Bean we will write under Springboot application mainclass with @SpringBootApplication

We need to define the shown @Bean configuration to set the CORS configuration support globally to your Spring Boot application.

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurerAdapter() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/products").allowedOrigins("http://localhost:9000");

}

};

}

**Versioning in SpringRest :**

**Versioning Strategies:**

1. Through URI path 2.versioning through query parameter 3. Versioning through custome headers 4.versioning through content negotiation

1st:

2nd: versioning Through query parameter example given :

@RestController

@RequestMapping(“/plans”)

**Public class PlanController**{

@AutoWired

Private PlanService planService;

@GetMapping(value=”/{planId}”,param=”version=1”)

**Public planDto fetchPlanById(@pathvariable(“planId”) int planId)**{  
return planService.fetchPlanById(planId); // it returns all plan that id can enroll .

}

@GetMapping(value=”/{planId}”,param=”version=2”)

**Public planDto fetchPlanByIdv2(@pathvariable(“planId”) int planId)**{  
return planService.fetchPlanByIdv2(planId); // it returns only plan associate with id

Uri =http://lolcalhost:8080/plans/210/?version=1

Uri =http://lolcalhost:8080/plans/210/?version=2

Note : Now this two version will produce different result . because with same palnId .

**3rd:** Versioning through custome headers -- > This type of versioning requires the inclusion of custom headers in the request URI to map to the correct version of the API Endpoint

@RestController

@RequestMapping(“/plans”)

**Public class PlanController**{

@AutoWired

Private PlanService planService;

@GetMapping(value=”/{planId}”,headers=”X-API-VERSION=1”)

**Public planDto fetchPlanById(@pathvariable(“planId”) int planId)**{  
return planService.fetchPlanById(planId);

}

Uri and header mentioning :

<http://localhost:8080/plans/201>

headers=[X-API-VERSION=2]

4th:  **Accept Header** Versioning or content negotiation -- > This type of versioning requires the inclusion of custom headers in the request URI to map to the correct version of the API Endpoint

@RestController

@RequestMapping(“/plans”)

**Public class PlanController**{

@AutoWired

Private PlanService planService;

@GetMapping(value=”/{planId}”, produces=”application/vnd.plans.app-v1+json”)

**Public planDto fetchPlanById(@pathvariable(“planId”) int planId)**{  
return planService.fetchPlanById(planId);

}-

Uri : <http://localhost:8080/plans/201>

Headers=[Accept=application/vnd.plans.app-v1+json]