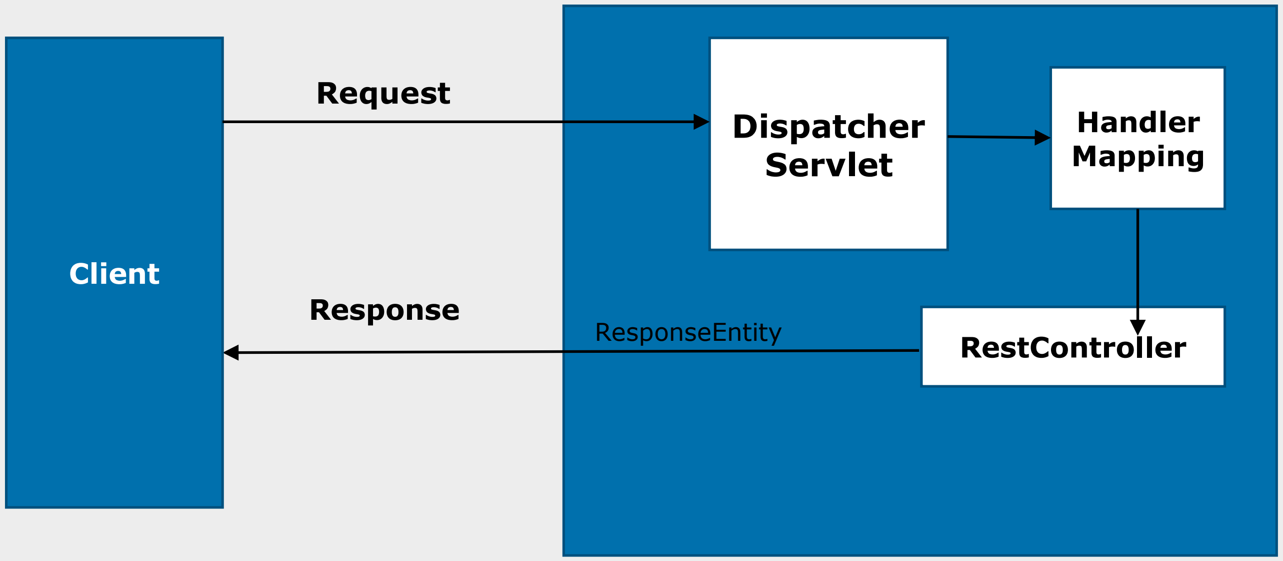
Spring RESTFul Web Services

Basic Spring 5.0

Lesson Objectives

- Spring MVC REST Workflow
- SpringREST Introduction
- Life cycle of a Request in Spring MVC Restful
- Why REST Controller ?
- HTTP methods in REST
- HTTP Status Code
- HTTP request Mapping
- RESTful URLs – HTTP methods
- @PathVariable, @RequestBody Annotation
- ResponseEntity Object
- Cross-Origin Resource Sharing (CORS)
- REST Testing
- Spring RestTemplate methods

6.1 Spring MVC REST Workflow



Spring5 MVC REST Workflow

6.2 Spring REST Introduction

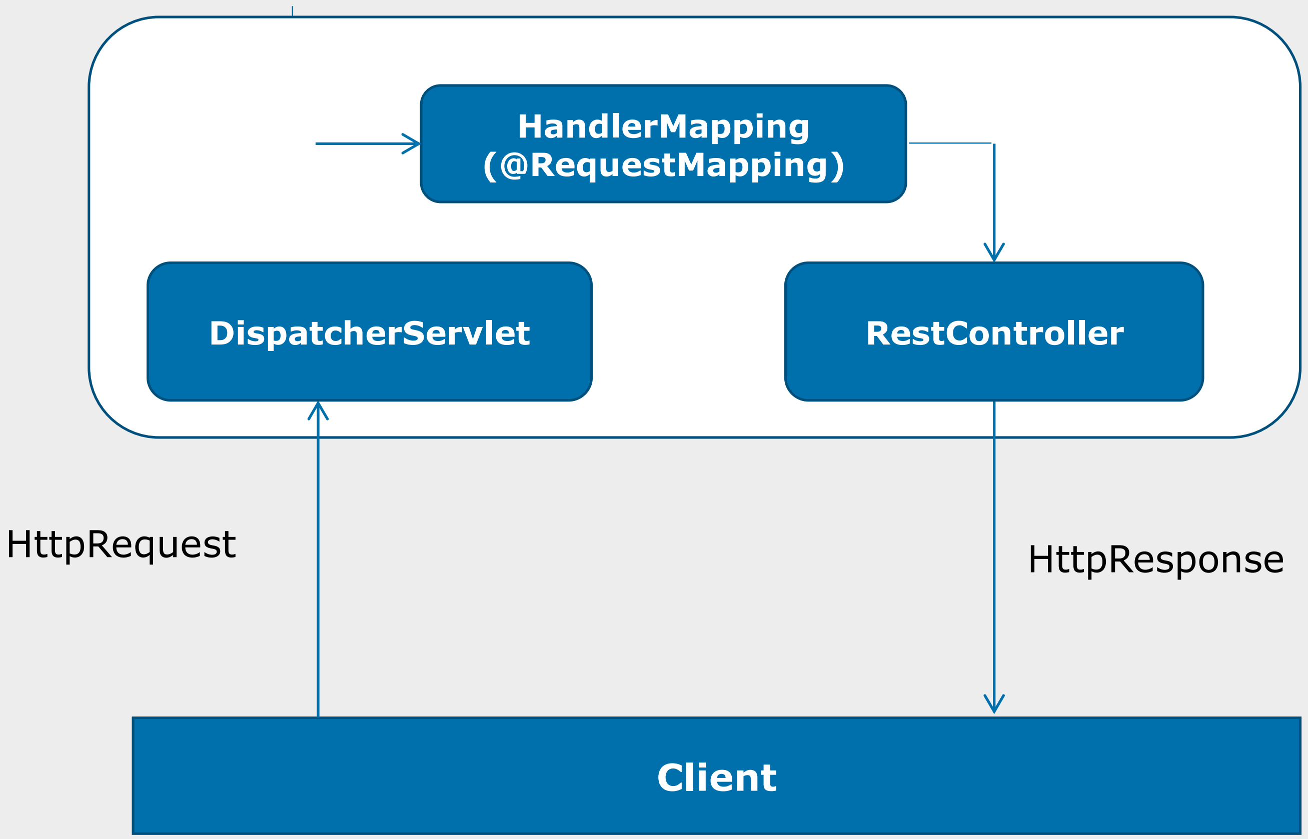
@RestController = @Controller + @ResponseBody

- RestController class should be annotated as @RestController
- Returns **JSON,XML,html,text,etc.,** from controller
- By default it returns JSON

```
@RestController
public class CountryController {

 @RequestMapping(value = "/countries", method =
RequestMethod.GET,headers="Accept=application/json")
 public List getCountries()
 {
  List listOfCountries = new ArrayList();
  listOfCountries=createCountryList();
  return listOfCountries;
 }
```

6.3 Life cycle of a Request in Spring MVC Restful



6.4 Why REST Controller ?

- Server should deal with only data.
- Popular front-end frameworks like Angular, EmberJs, ReactJs etc.,
- Expose the data as JSON/XML/HTML/plain text.
- Server can process business logic quickly
- Server no need to produce presentation tier.
- Elimination of JSP from current Architecture.
- Leads API-led connectivity Architecture
- Encourage microservices kind of application in enterprise architecture.

6.5 HTTP methods in REST

HTTP Method	Operation	Comment
GET	Read Operation only	Uses only for the read operation.GET should be idempotent
POST	Create new resource	Should only be used to create a new resource
PUT	Update / Replace Resource	Update an existing resource.Think of PUT method as putting a resource
DELETE	Delete Resource	To remove a given resource.DELETE operation is *idempotent*
PATCH	Partial Update / Modify	Partial update to a resource should happen through PATCH

6.6 HTTP Status Code

Status Code Category	Description	Example
1XX – Informational	Informational indicates a provisional response	100 (Continue) , 101
2XX – Successful	This class of status code indicates that the client’s request was successfully received, understood, and accepted.	200 (OK), 201(Created), 202 (Accepted)
3XX – Redirection	This class of status code indicates that further action required by the user agent to fulfill the request	301 (Moved Permanently), 302, 304
4XX – Client Error	The 4xx class of status code is intended for cases where the client seems to have erred	400 (Bad Request), 401, 403, 404, 405
5XX – Server Error	Response status codes beginning with the digit “5” tell cases where the server is aware that it has erred or is incapable of performing the request	500 (Internal Server Error), 502, 503, 505

6.6 HTTP Status Code

- 200 - OK
- 201 - Created
- 202 - Accepted
- 304 - Not Modified
- 400 - Bad Request
- 401 - Unauthorized
- 403 - Forbidden
- 404 - Not Found

6.7 HTTP request mapping

@RequestMapping

@GetMapping

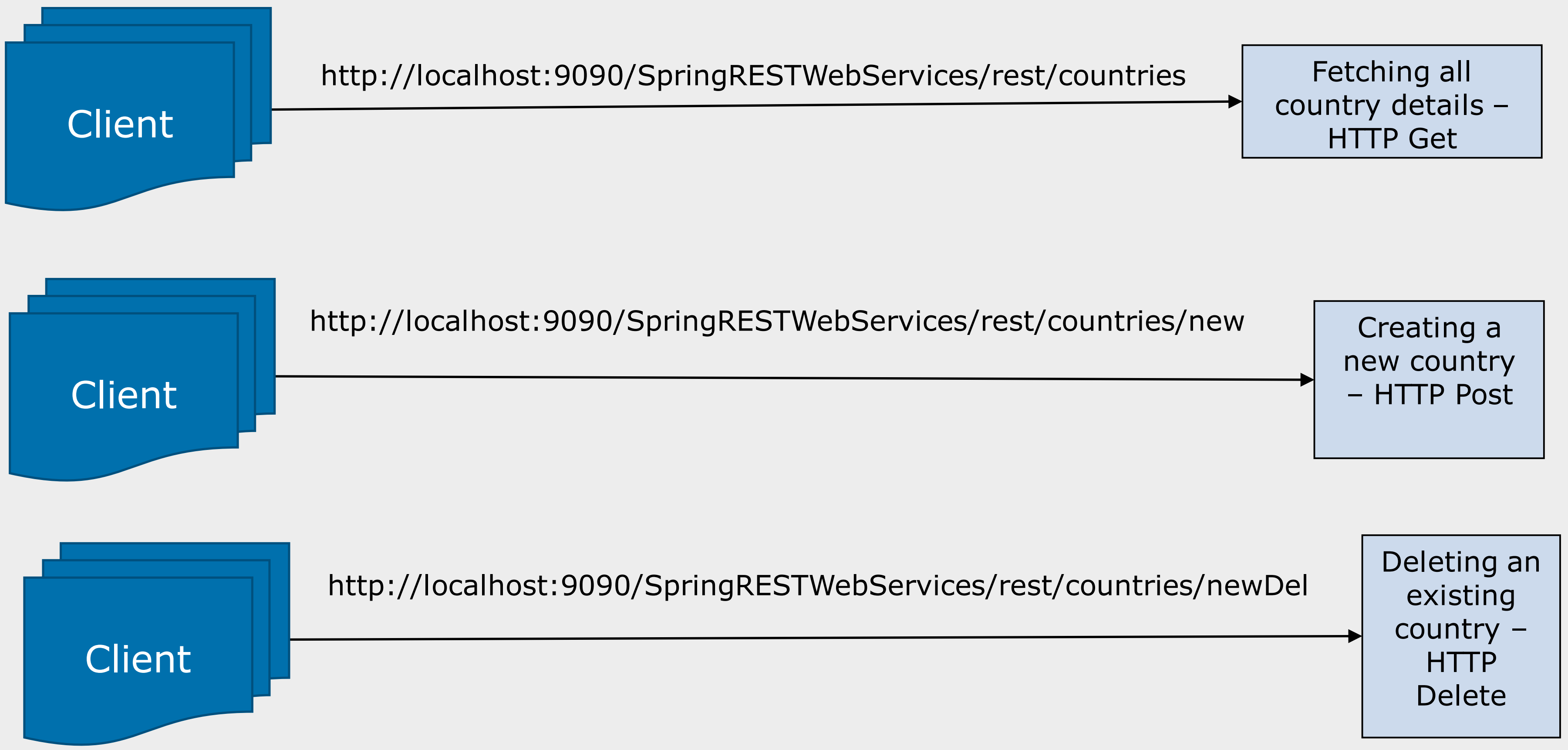
@PostMapping

@PutMapping

@DeleteMapping

@PatchMapping

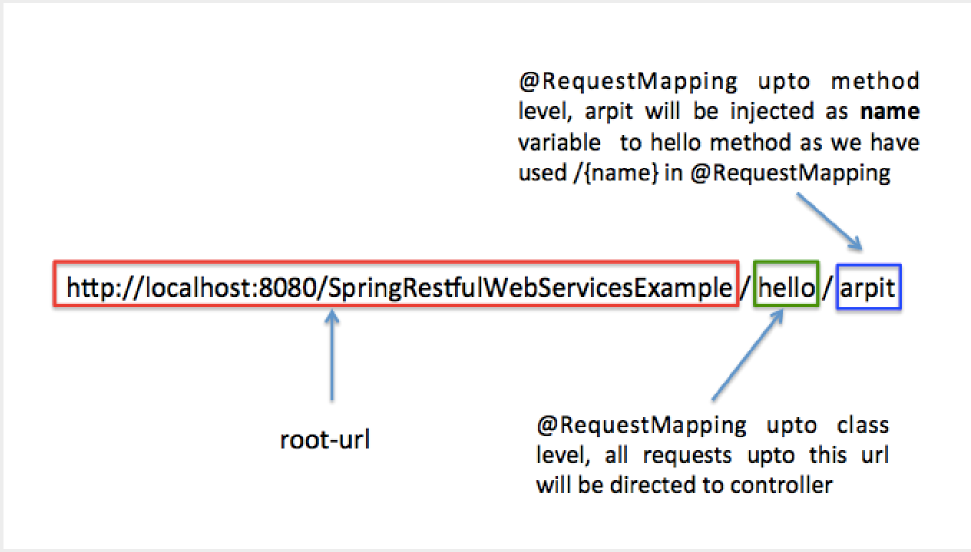
6.8 RESTful URLs – HTTP methods


Client
http://localhost:9090/SpringRESTWebServices/rest/countries
Fetching all country details – HTTP Get
Client
http://localhost:9090/SpringRESTWebServices/rest/countries/new
Creating a new country – HTTP Post
Client
http://localhost:9090/SpringRESTWebServices/rest/countries/newDel
Deleting an existing country – HTTP Delete

6.9 @PathVariable Annotation

Used to inject values from the URL into a method parameter. This way you inject name in hello method .

```
@RestController
@RequestMapping("/hello")
public class SpringRestController {
@RequestMapping(value = "/{name}",
                method = RequestMethod.GET)
public String hello(@PathVariable String name) {
                String result="Hello "+name;
                return result;
}
}
```


@RequestMapping upto method level, arpit will be injected as **name** variable to hello method as we have used /{name} in @RequestMapping
http://localhost:8080/SpringRestfulWebServicesExample / hello / arpit
root-url
@RequestMapping upto class level, all requests upto this url will be directed to controller

6.9 ResponseEntity Object

- ResponseEntity is a **generic type**
- Used to **Manipulate** the HTTP Response
- Represents the whole HTTP response: **status code, headers, and body**
- provides two nested builder interfaces: **HeadersBuilder** and its subinterface, **BodyBuilder**
- **Alternate** for ResponeEntity - @ResponseBody, @ResponseStatus and HttpServletResponse

```
@GetMapping("/hello")
ResponseEntity<String> hello() {
    return new ResponseEntity<>("Hello World!", HttpStatus.OK);
}
```

6.9 @RequestBody annotation

- If a method parameter is annotated with @RequestBody, Spring will bind the incoming HTTP request body(for the URL mentioned in @RequestMapping for that method) to that parameter.
- While doing that, Spring will use HTTP Message converters to convert the HTTP request body into domain object **[deserialize request body to domain object]**, based on **Accept** header present in request.

```
@RestController
public class EmployeeController {
@Autowired
IEmployeeService empservice;
@RequestMapping(value ="/employee/create/", consumes = MediaType.APPLICATION_JSON_VALUE,
headers="Accept=application/json",method = RequestMethod.POST)
public List<Employee> createEmployee(@RequestBody Employee emp) {

            empservive.addEmployee(emp);
            return empservice.getAllEmployee();
} }
```

6.10 Cross-Origin Resource Sharing (CORS)

- **CORS** (Cross-origin resource sharing) allows a webpage to request additional resources into browser from **other domains** e.g. fonts, CSS or static images from CDNs.
- Helps in serving web content from multiple domains into browsers who usually have the same-origin security policy.
- **Spring CORS** support in Spring MVC application at method level and global level.
- **@CrossOrigin** allows all origins, all headers, the HTTP methods specified in the **@RequestMapping** annotation and a **maxAge of 30 minutes**.

6.10 @CrossOrigin Annotation Attributes

- **Origins** - List of allowed origins. It's value is placed in the Access-Control-Allow-Origin header of both the pre-flight response and the actual response.
    - – * – means that all origins are allowed.
    - – If undefined, all origins are allowed.
- **allowedHeaders** - List of request headers that can be used during the actual request. Value is used in preflight's response header Access-Control-Allow-Headers.
    - – * – means that all headers requested by the client are allowed.
    - – If undefined, all requested headers are allowed.
- **methods** - List of supported HTTP request methods. If undefined, methods defined by RequestMapping annotation are used.
- **exposedHeaders** - List of response headers that the browser will allow the client to access. Value is set in actual response header Access-Control-Expose-Headers.
    - – If undefined, an empty exposed header list is used.

6.10 @CrossOrigin Annotation Attributes

- **allowCredentials** - It determine whether browser should include any cookies associated with the request.

  – false – cookies should not included.

  – "" (empty string) – means undefined.

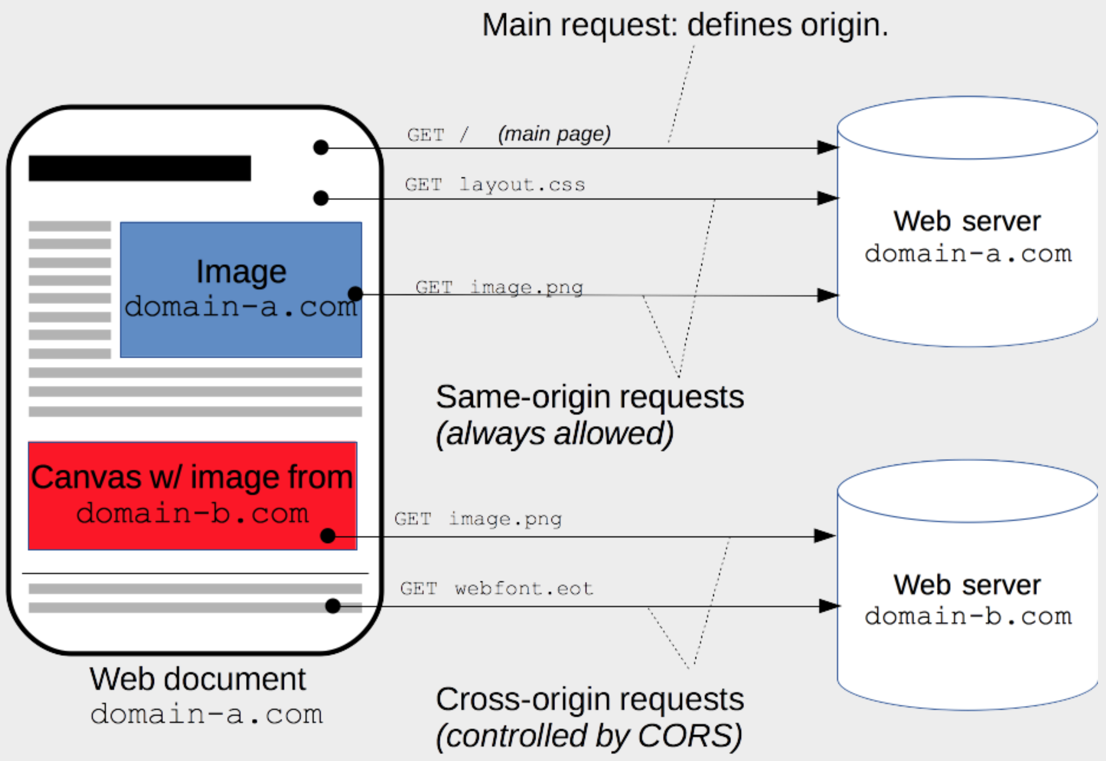
  – true – pre-flight response will include the header Access-Control-Allow-Credentials with value set to true.

  – If undefined, credentials are allowed.

- **maxAge** - maximum age (in seconds) of the cache duration for pre-flight responses. Value is set in header Access-Control-Max-Age.

  – If undefined, max age is set to 1800 seconds (30 minutes).

Cross-Origin Resource Sharing (CORS)



Cross-Origin Resource Sharing (CORS)

```
@CrossOrigin(origins = "http://localhost:4200")
@RestController
public class CountryController {
@Autowired
private ICountryService service;

//@CrossOrigin(origins = "http://localhost:4200")
@RequestMapping(value = "/countries/search/{id}",method =
RequestMethod.GET,headers="Accept=application/json")
public Country getCounty(@PathVariable int id) {
return service.searchCountry(id);
}
}
```

6.11 REST Testing

Spring RestTemplate

- Spring RestTemplate class is part of spring-web, introduced in Spring 3.
- We can use **RestTemplate to test HTTP based restful web services**, it doesn’t support HTTPS protocol.
- RestTemplate class provides overloaded methods for different HTTP methods, such as GET, POST, PUT, DELETE etc.

URI	HTTP METHOD	DESCRIPTION
/springData/person	GET	Get all persons from database
/springData/person/{id}	GET	Get person by id
/springData/person	POST	Add person to database
/springData/person	PUT	Update person
/springData/person/{id}	DELETE	Delete person by id

6.12 Spring RestTemplate Methods

Get:

getForObject, getForEntity

Post:

postForObject(String url, Object request, Class responseType, String... uriVariables) postForLocation(String url, Object request, String... urlVariables),

Put:

put(String url, Object request, String...urlVariables)

Delete:

delete()

Head:

headForHeaders(String url, String... urlVariables)

Options:

optionsForAllow(String url, String... urlVariables)

Demo: SpringRESTDemos

SpringRESTDemo

Summery

- Spring MVC REST Workflow
- SpringREST Introduction
- Life cycle of a Request in Spring MVC Restful
- Why REST Controller ?
- HTTP methods in REST
- HTTP Status Code
- HTTP request Mapping
- RESTful URLs – HTTP methods
- @PathVariable, @RequestBody Annotation
- ResponseEntity Object
- Cross-Origin Resource Sharing (CORS)
- REST Testing
- Spring RestTemplate methods

Spring ReSTful

Lab

Lab 2

Review Question

Question 1: How to access URI parameters in Spring REST?

- @RequestParam
- @QueryParam
- @PathVariable
- @ResponseParam

Question 2: ____________ is used to test RESTful API in Spring framework?

- RestTemplate
- RestAPITemplate
- Junit
- JQuery

Review Question

Question 3: ____________ specifies a media type a resource can generate.

- @PUT
- @POST
- @Produces
- @Consumes