Instructor Notes:

Add instructor notes here.

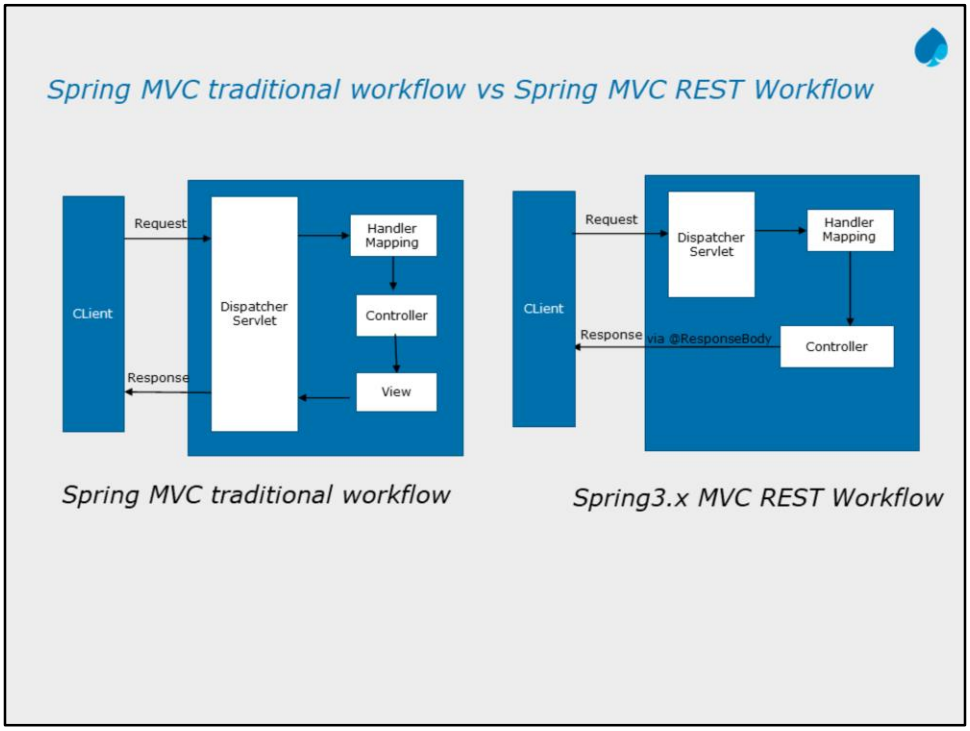
Instructor Notes:

Lesson Objectives

Introduction to Spring MVC framework

- *Spring MVC traditional workflow vs Spring MVC REST Workflow*
- Using the @ResponseBody Annotation
- Life cycle of a Request in Spring MVC Restful
- Why REST Controller ?
- *Spring 4.x MVC RESTful Web Services Workflow*
- REST Controller
- RESTful URLs – HTTP methods
- Cross-Origin Resource Sharing (CORS)
- @RequestBody annotation

Instructor Notes:



Spring's annotation based MVC framework simplifies the process of creating RESTful web services. The key difference between a traditional Spring MVC controller and the RESTful web service controller is the way the HTTP response body is created. While the traditional MVC controller relies on the View technology, the RESTful web service controller simply returns the object and the object data is written directly to the HTTP response as JSON/XML

Spring MVC REST Workflow

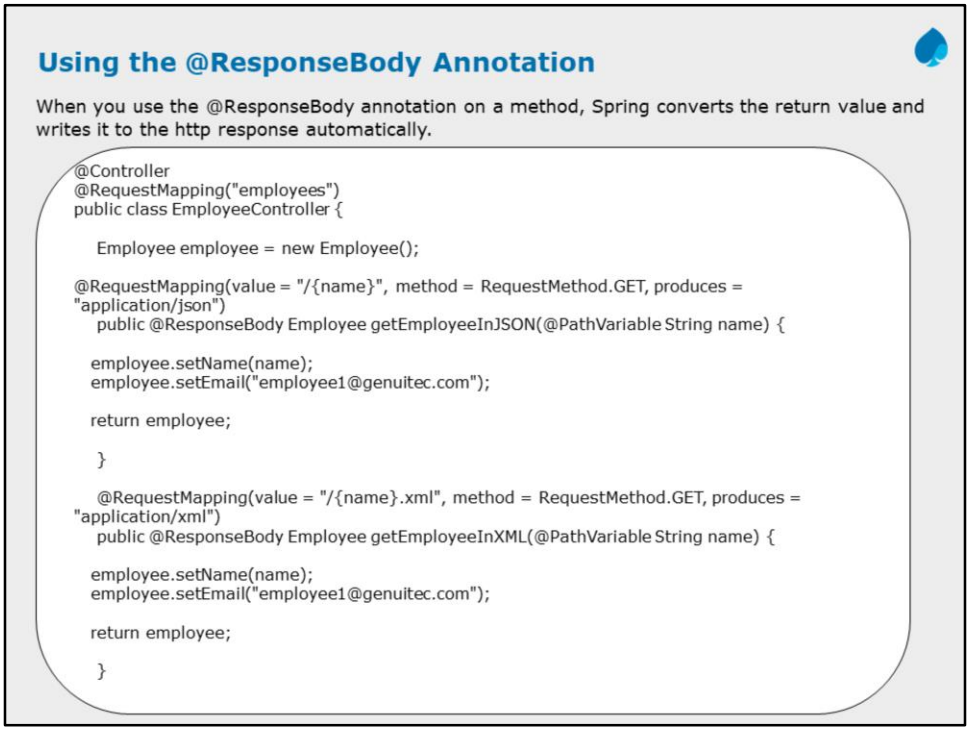
The following steps describe a typical Spring MVC REST workflow:
The client sends a request to a web service in URI form.
The request is intercepted by the DispatcherServlet which looks for Handler Mappings and its type.

• The Handler Mappings section defined in the application context file tells DispatcherServlet which strategy to use to find controllers based on the incoming request.

• Spring MVC supports three different types of mapping request URIs to controllers: annotation, name conventions and explicit mappings.

Requests are processed by the Controller and the response is returned to the DispatcherServlet which then dispatches to the view.

Instructor Notes:

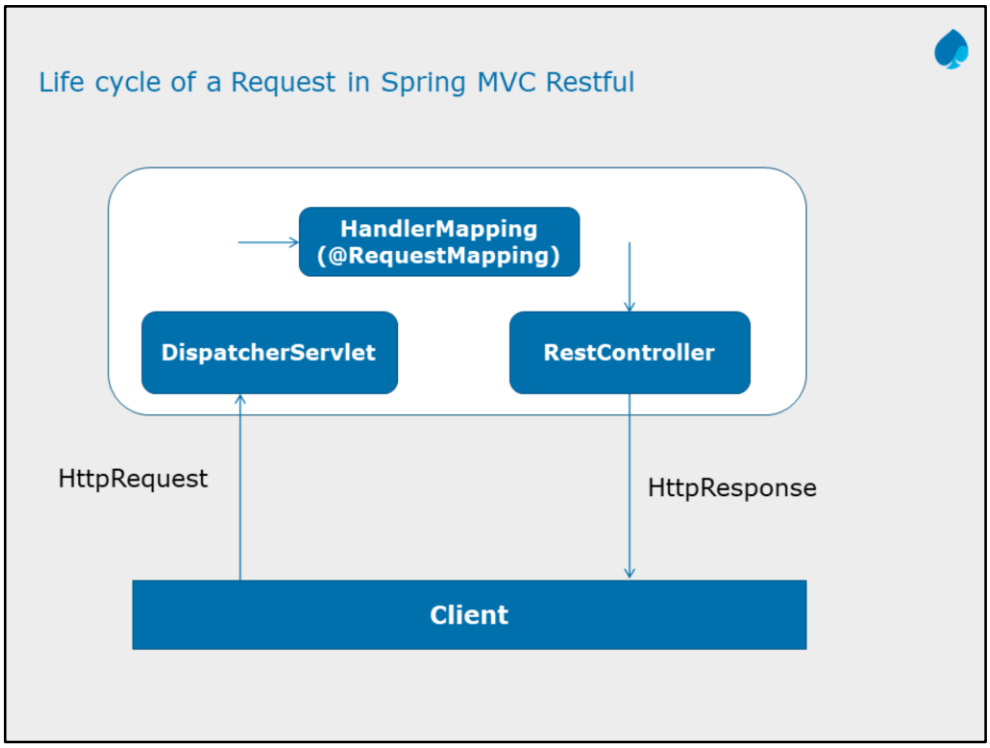


Spring has a list of HttpMessageConverters registered in the background. The responsibility of the HTTPMessageConverter is to convert the request body to a specific class and back to the response body again, depending on a predefined mime type. Every time an issued request hits @ResponseBody, Spring loops through all registered HTTPMessageConverters seeking the first that fits the given mime type and class, and then uses it for the actual conversion.

Instructor Notes:

Slide demonstrates Life cycle of Spring RESTful services.

Along with that also explains why Rest controllers were introduced.



REST(Representational State Transfer) is an architectural style with which Web Services can be designed that serves resources based on the request from client. A Web Service is a unit of managed code, that can be invoked using HTTP requests. You develop the core functionality of your application, deploy it in a server and expose to the network. Once it is exposed, it can be accessed using URI's through HTTP requests from a variety of client applications. Instead of repeating the same functionality in multiple client (web, desktop and mobile) applications, you write it once and access it in all the applications.

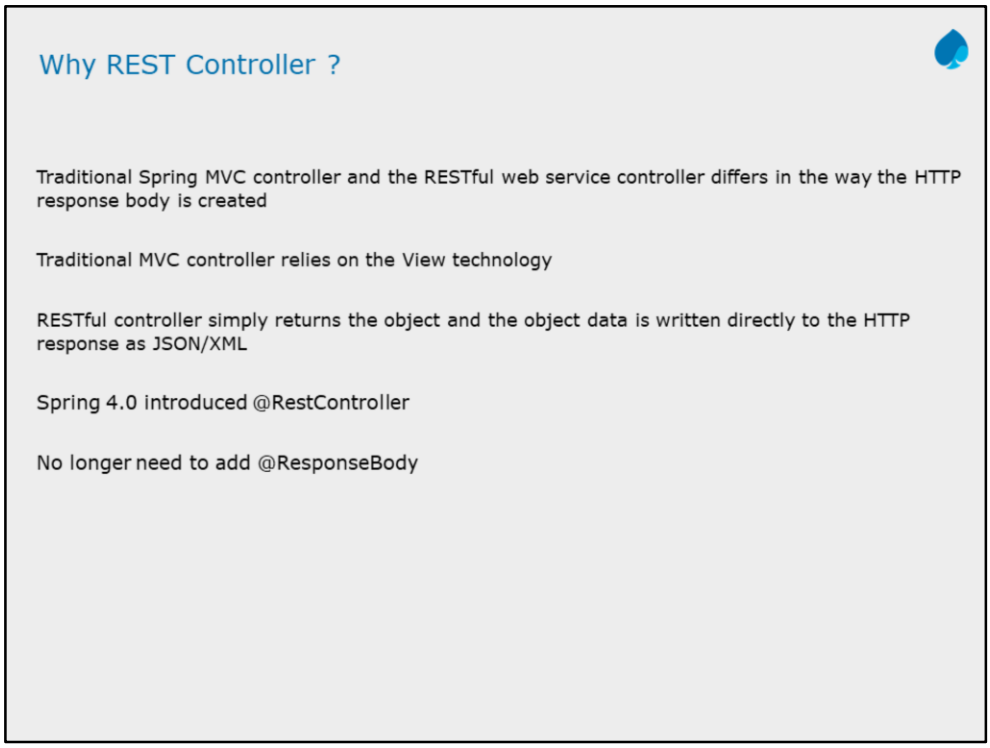
In the above diagram, from the time that a request is received by Spring until the time that a response is returned to the client, many pieces of Spring Restful webservices are involved.

The process starts when a client (typically a web browser) sends a request. It is first received by a DispatcherServlet. Like most Java-based MVC frameworks, Spring MVC uses a front-controller servlet (here DispatcherServlet) to intercept requests. This in turn delegates responsibility for a request to other components of an application for actual processing.

The Spring MVC uses a Controller component for handling the request. But a typical application may have several controllers. To determine which controller should handle the request, DispatcherServlet starts by querying one or more HandlerMappings. A HandlerMapping typically maps URL patterns to RestControllers.

Once the DispatcherServlet has a appropriate RestController selected, it dispatches the request to that Controller which performs the business logic (a well-designed RestController object delegates responsibility of business logic to one or more service objects). Upon completion of business logic, HTTPResponse is generated and sent back to the client.

Instructor Notes:



Spring 4.0 introduced @RestController, a specialized version of the controller which is a convenience annotation that does nothing more than add the @Controller and @ResponseBody annotations. By annotating the controller class with @RestController annotation, you no longer need to add @ResponseBody to all the request mapping methods. The @ResponseBody annotation is active by default

The key difference between a traditional Spring MVC controller and the RESTful web service controller is the way the HTTP response body is created. While the traditional MVC controller relies on the View technology, the RESTful controller simply returns the object and the object data is written directly to the HTTP response as JSON/XML.

In Spring 3 @ResponseBody was used as an annotation over the Rest Controller methods on the return type indicating the HTTP response as JSON/XML, along with this we had to specify the MIME type as “application/json” or “application/xml”.

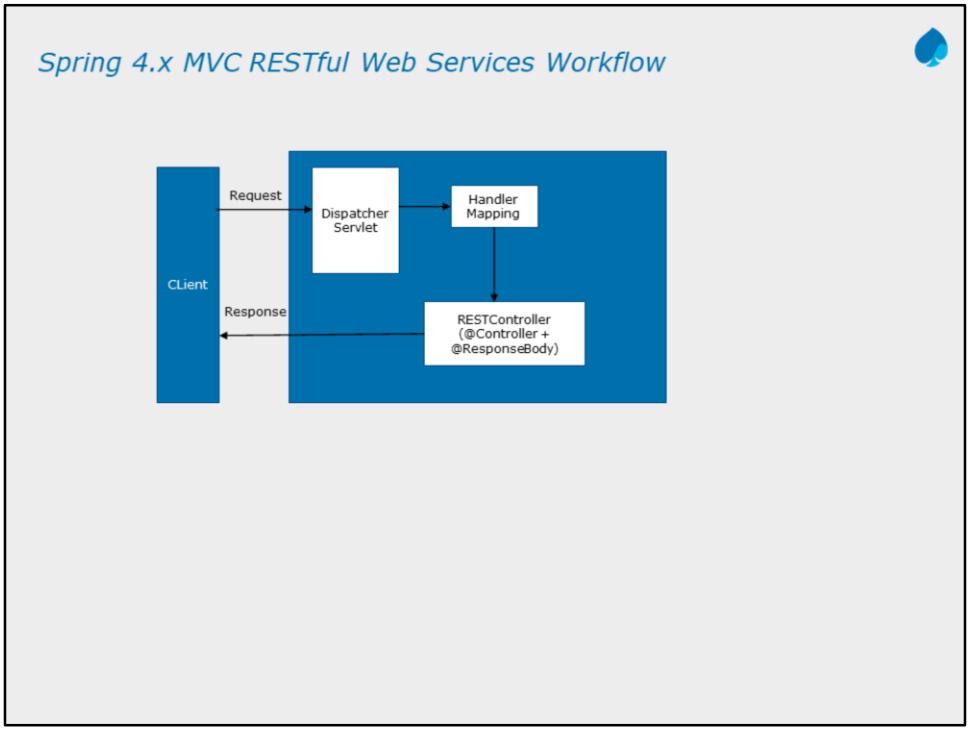
For example: Spring 3 Rest Controller method:

```
(
@Controller
@RequestMapping("employees")
public class EmployeeController {

@RequestMapping(value = "/{name}", method = RequestMethod.GET, produces =
"application/json")
    public @ResponseBody Employee getEmployeeInJSON(@PathVariable String name) {
                        employee.setName(name);
                        employee.setEmail("employee1@info.com");
                        return employee;
    }
)
```

In Spring 4 @RestController is a combination of @Controller + @ResponseBody annotations. Thus in Spring 4 we do not need to use @ResposneBody; we directly can use @RestController and return view and or the object to the HTTP response.

In our example discussed in next subsequent slides we have kept view (jsp) itself as the ResponseBody.

Instructor Notes:



Spring 4.0 introduced @RestController, a specialized version of the controller which is a convenience annotation that does nothing more than add the @Controller and @ResponseBody annotations. By annotating the controller class with @RestController annotation, you no longer need to add @ResponseBody to all the request mapping methods. The @ResponseBody annotation is active by default.

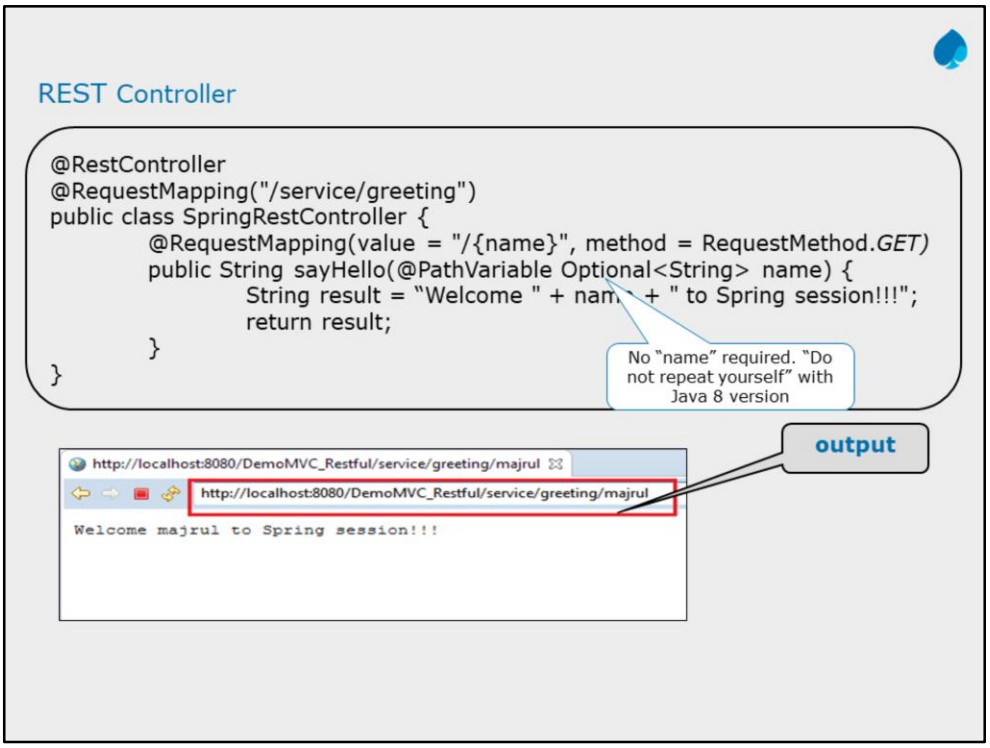
Instructor Notes:

Slide introduces Spring 4 REST concepts with HTTP methods

Spring supports different Request methods like GET, PUT, DELETE, POST corresponding to respective HTTP methods.

Instructor Notes:

Slide demonstrates how a RestController can be created and how to use @PathVariable



Let us see what a RESTful controller looks like.

Restful controller @RequestMapping annotation which says that this controller will handle requests for /service/greeting. It implies and supports the fact that this controller is focused on displaying greeting message.

URL (uniform resource locator) is a means of locating a resource. But, since, no two resources can share the same URL, URL could also be a means of uniquely identifying a resource ie URI. Many URLs don't locate or identify anything—they make demands.

Rather than identify something, they demand some action be taken. For example consider below URL:

http://localhost:8080/DemoMVC_Restful/service/greeting?name=capgemini, this kind of URL will be handled by the SpringRestController's sayHello() method. Here, the URL is not locating or identifying a resource. The base portion of the URL is verb-oriented ie "sayXXX()".

RESTful URLs on the other hand are resource-oriented. So, the URL in the example above should look like the following :
http://localhost:8080/DemoMVC_Restful/service/greeting/capgemini

This URL now correctly locates and identifies a resource. Notice, the base portion of the URL is resource-oriented. Also notice that it has no query parameters, instead it has a parameterized path. The RESTful URL's input is part of the URL's path.

But how will you code the controller to take input from the URL's path?

At the method level, {name} in the @RequestMapping annotation, is a placeholder through which variable data will be pass into the method.

The sayHello() method is designed to handle GET requests for URLs that take the form / {name}. The @PathVariable annotation corresponds to this id.
So, if a GET request comes in for
http://localhost:8080/DemoMVC_Restful/service/greeting/capgemini, the sayHello() method will be called with capgemini passed in for the name parameter.

The method then uses that value to generate welcome message and place it into the model.

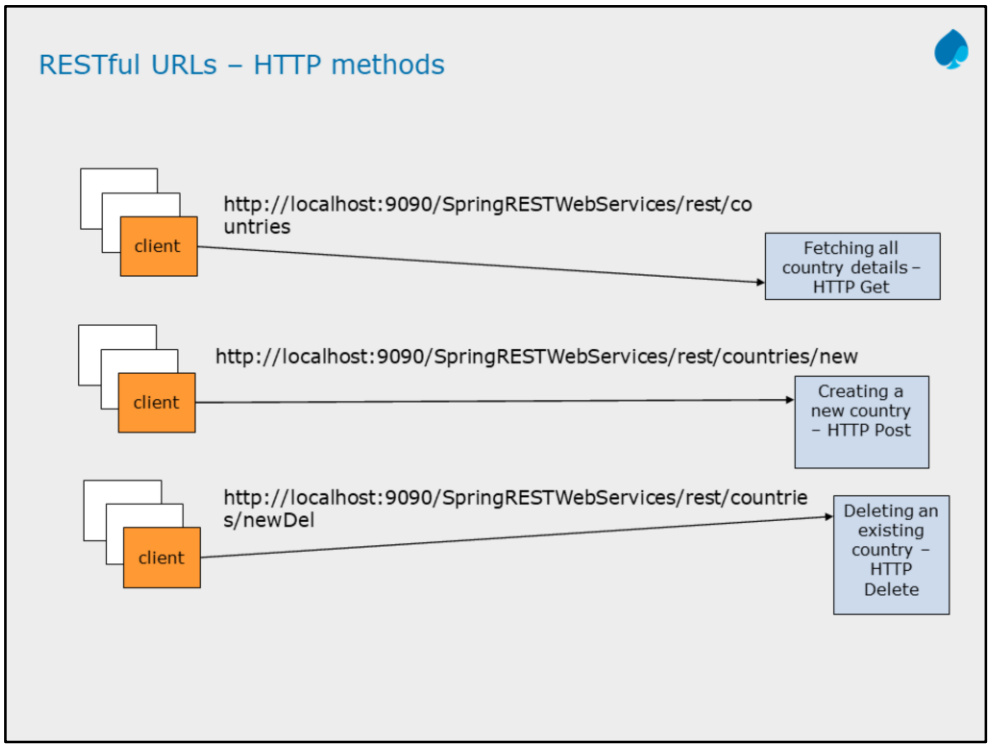
Instructor Notes:

REST Controller

```
@RestController
@RequestMapping("employees")
public class EmployeeController {

    Employee employee = new Employee();

    @RequestMapping(value = "/{name}", method = RequestMethod.GET, produces =
"application/json")
    public Employee getEmployeeInJSON(@PathVariable String name) {

   employee.setName(name);
   employee.setEmail("employee1@genuitec.com");

   return employee;

    }

    @RequestMapping(value = "/{name}.xml", method = RequestMethod.GET, produces =
"application/xml")
    public Employee getEmployeeInXML(@PathVariable String name) {

   employee.setName(name);
   employee.setEmail("employee1@genuitec.com");

   return employee;

    }

}
```

Instructor Notes:

Trainer can explain concept of URL mapping to different HTTP methods and notes page by referring the demo code shared : SpringRESTWebServices



The slide demonstrates different RESTful URLs with respect to different HTTP methods.

Demo: SpringRESTWebServices can be used as a reference.

Note: At times if the HTTP Get is used over a couple of RestController methods it has to be combined with URL patterns to create unique identifications.

1. In the above slide first URL pattern demonstrates : HTTP Get method to fetch all country details.

Similarly if details for a particular country need to be fetched then the country id can be appended in URL and extracted via the @PathVariable

http://localhost:9090/SpringRESTWebServices/rest/countries/3 -> With this URL country details are fetched for country Id = 3

2. The second URL pattern demonstrates : HTTP Post method to create a new country

3. The third URL pattern demonstrates : HTTP Delete method to delete an existing country

Note: As HTML supports only Get and Post methods for the method attribute in the form tag; we also need to map the HTTP PUT(update) and HTTP DELETE (delete) methods to update and delete the resources respectively.

For this Spring provides us with a Filter-mapping which is to be given in web.xml file:

```
<filter>
   <filter-name>httpMethodFilter</filter-name>
   <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
 </filter>
 <filter-mapping>
   <filter-name>httpMethodFilter</filter-name>
   <servlet-name>dispatcher</servlet-name>
 </filter-mapping>
```

By using this Spring in-built filter the different methods of HTTP specification will be mapped to their actual HTTP implementations.
Here the filter will be intercepted for all the requests coming to DispatcherServlet.

Also in the JSP pages for updating and deleting a country need to pass a HTML hidden parameter **:** For example

<input type="hidden" name="_method" value="delete"/> to pass the "real" HTTP method to Spring Rest Controller.

Instructor Notes:

Cross-Origin Resource Sharing (CORS)

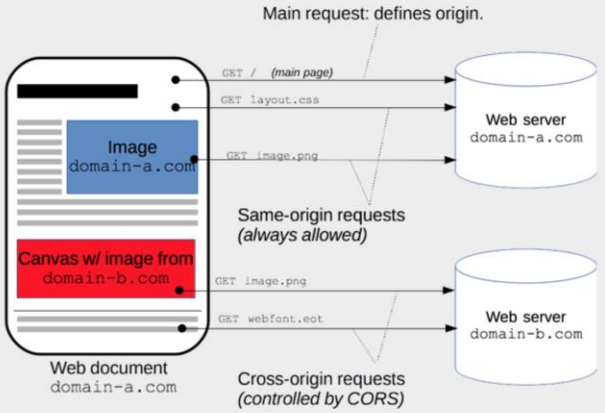
- Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to let a user agent gain permission to access selected resources from a server on a different origin (domain) than the site currently in use.
- A user agent makes a **cross-origin HTTP request** when it requests a resource from a different domain, protocol, or port than the one from which the current document originated.
- An example of a cross-origin request: A HTML page served from http://domain-a.com makes an <img> src request for http://domain-b.com/image.jpg. Many pages on the web today load resources like CSS stylesheets, images, and scripts from separate domains, such as content delivery networks (CDNs)

Reference : https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Instructor Notes:

Cross-Origin Resource Sharing (CORS)

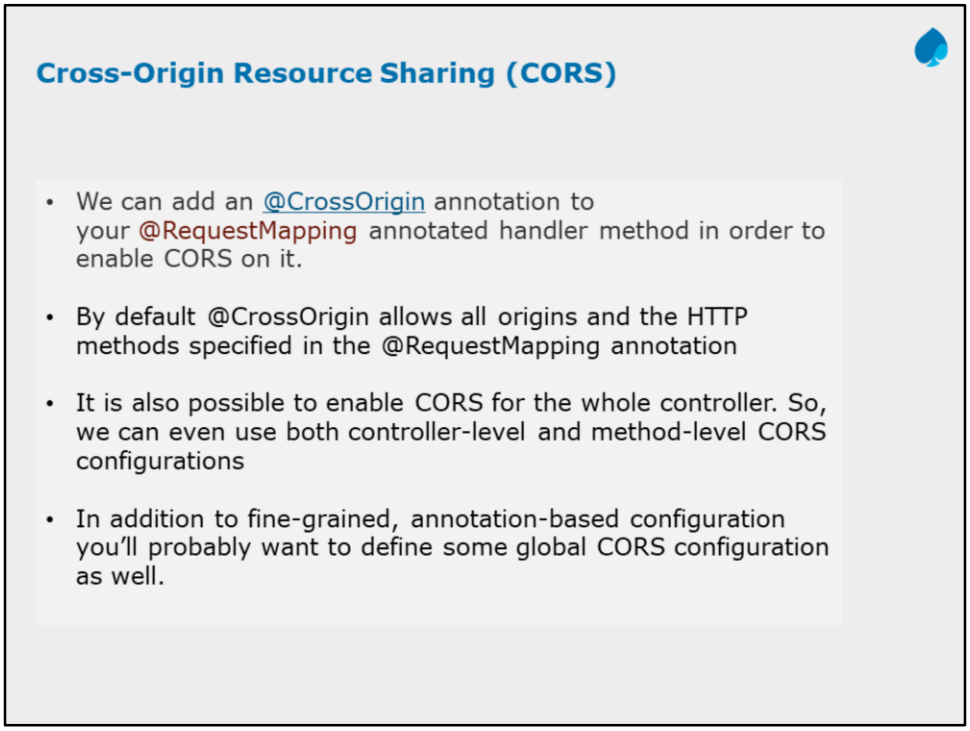
For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts. For example, XMLHttpRequest and the Fetch API follow the same-origin policy. This means that a web application using those APIs can only request HTTP resources from the same domain the application was loaded from unless CORS headers are used.



Reference : https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

The CORS mechanism supports secure cross-domain requests and data transfers between browsers and web servers. Modern browsers use CORS in an API container such as XMLHttpRequest or Fetch to help mitigate the risks of cross-origin HTTP requests.

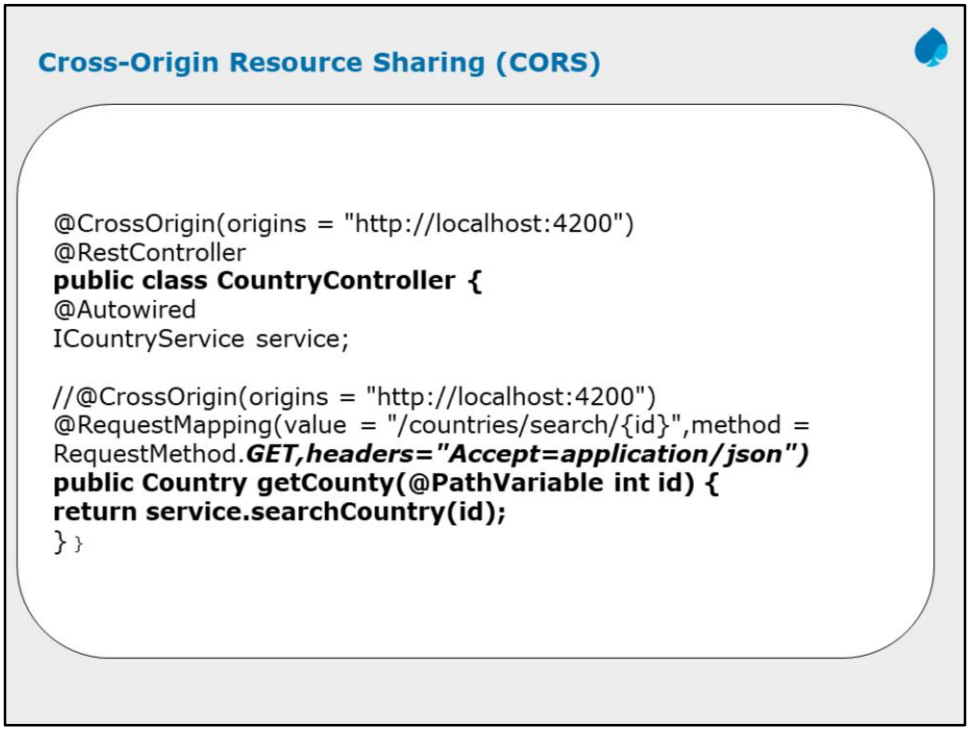
Instructor Notes:



Reference : http://javasampleapproach.com/spring-framework/spring-boot/spring-cors-example-crossorigin-spring-boot

– **origins**: specifies the URI that can be accessed by resource. "*" means that all origins are allowed. If undefined, all origins are allowed.
– **allowCredentials**: defines the value for **Access-Control-Allow-Credentials** response header. If value is true, response to the request can be exposed to the page. The credentials are cookies, authorization headers or TLS client certificates. The default value is **true**.
– **maxAge**: defines maximum age (in seconds) for cache to be alive for a pre-flight request. By default, its value is 1800 seconds.
We also have some attributes:
– **methods**: specifies methods (GET, POST,…) to allow when accessing the resource. If we don't use this attribute, it takes the value of @RequestMapping method by default. If we specify methods attribute value in @CrossOrigin annotation, default method will be overridden.
– **allowedHeaders**: defines the values for **Access-Control-Allow-Headers** response header. We don't need to list headers if it is one of *Cache-Control*, *Content-Language*, *Expires*, *Last-Modified*, or *Pragma*. By default all requested headers are allowed.
– **exposedHeaders**: values for **Access-Control-Expose-Headers** response header. Server uses it to tell the browser about its whitelist headers. By default, an empty exposed header list is used.

Instructor Notes:



Instructor Notes:

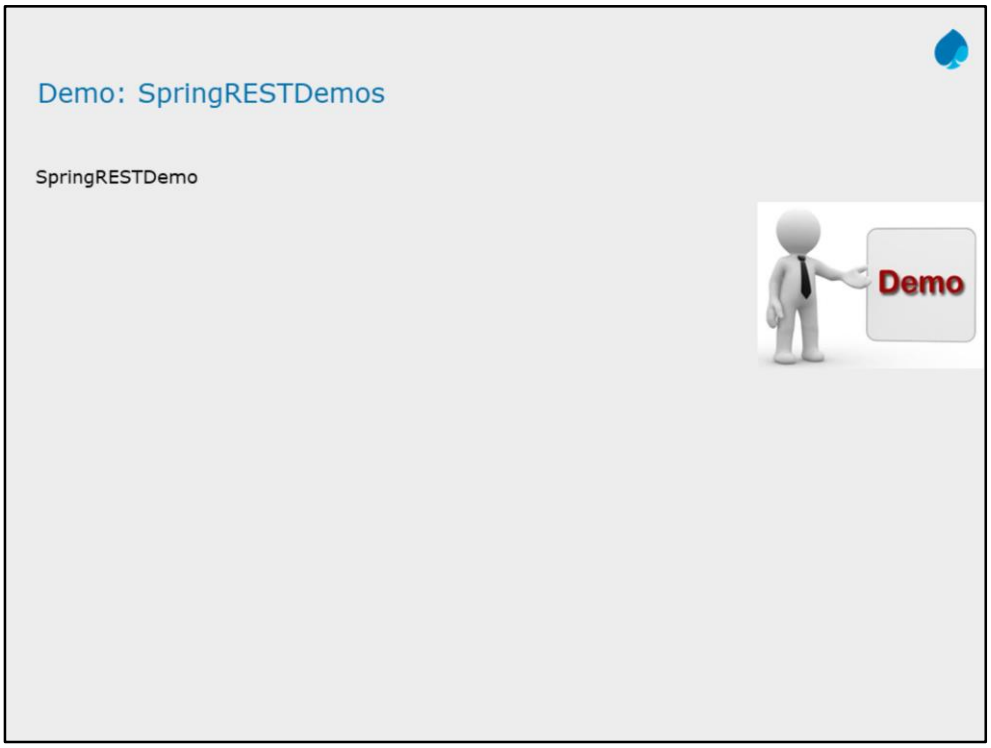
@RequestBody annotation

- 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();
}}
```

Instructor Notes:

These demos can be executed for better understanding



Instructor Notes:

Summery

We have so far learnt

- *Spring MVC traditional workflow vs Spring MVC REST Workflow*
- Using the @ResponseBody Annotation
- Life cycle of a Request in Spring MVC Restful
- Why REST Controller ?
- *Spring 4.x MVC RESTful Web Services Workflow*
- REST Controller
- RESTful URLs – HTTP methods

References:
https://www.genuitec.com/spring-frameworkrestcontroller-vs-controller/

Instructor Notes:

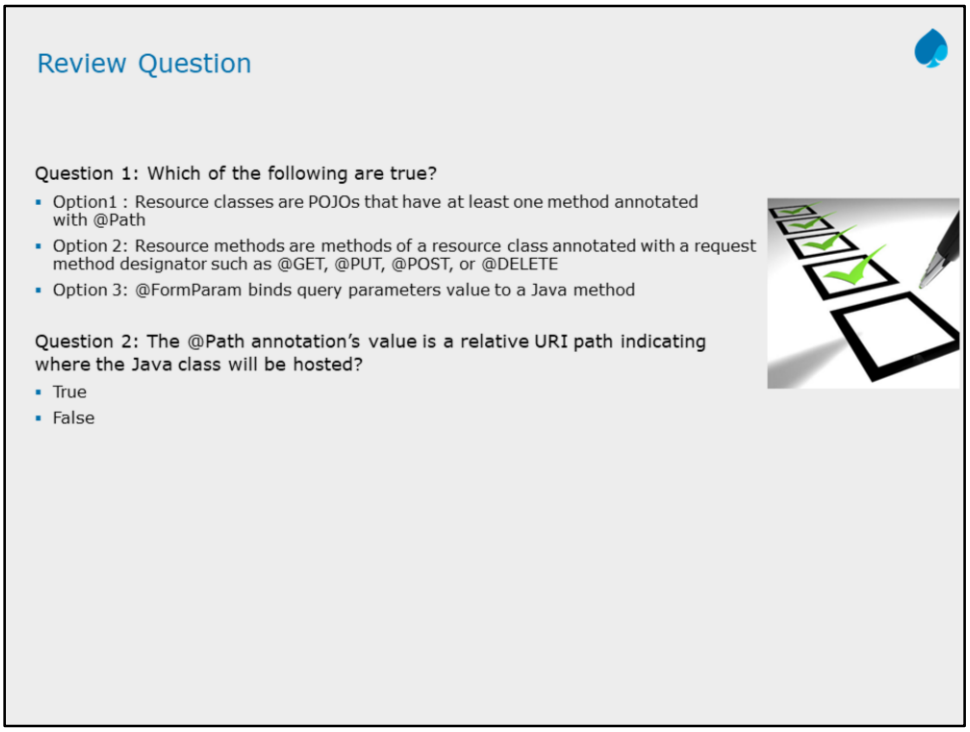
Corresponding lab assignment

Add the notes here.

Instructor Notes:

Question 1: Option 1, Option 2

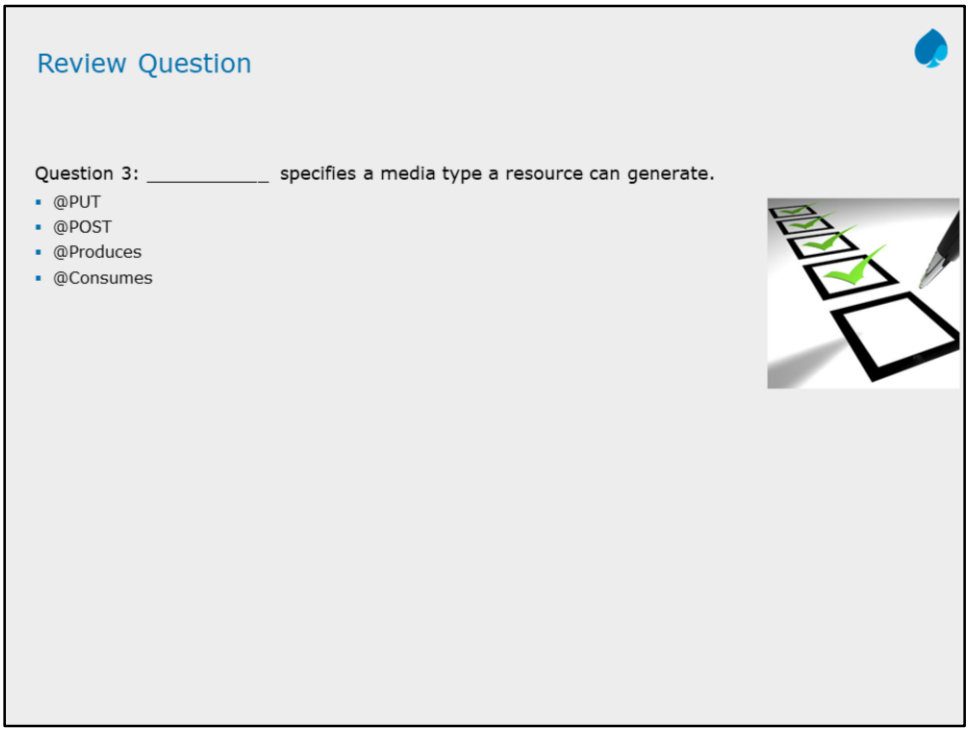
Question 2: True



Add the notes here.

Instructor Notes:

Question 3: @Produces



Add the notes here.