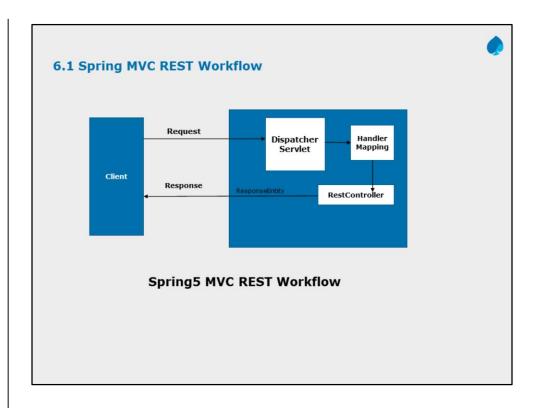
Add instructor notes here.



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





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

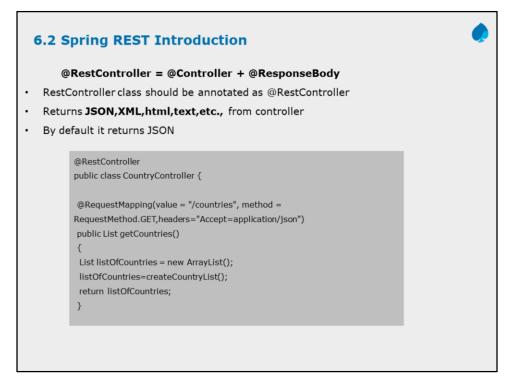
Spring MVC REST Workflow

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.

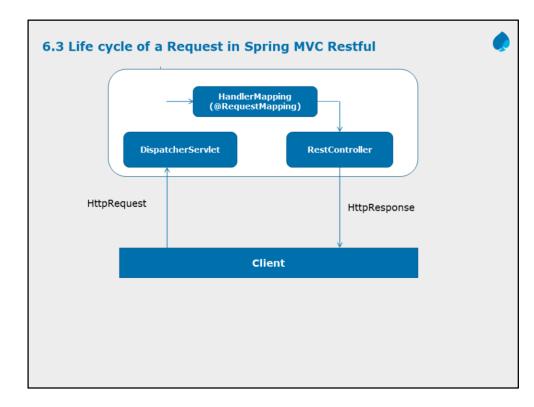
- The Handler Mappings section defined in the application context file tells DispatcherServlet which strategy to use to find controllers based on the incoming request.
- Spring MVC supports three different types of mapping request URIs to controllers: annotation, name conventions and explicit mappings. Requests are processed by the Controller and the response is returned to the DispatcherServlet which then dispatches to the view.



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.

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.

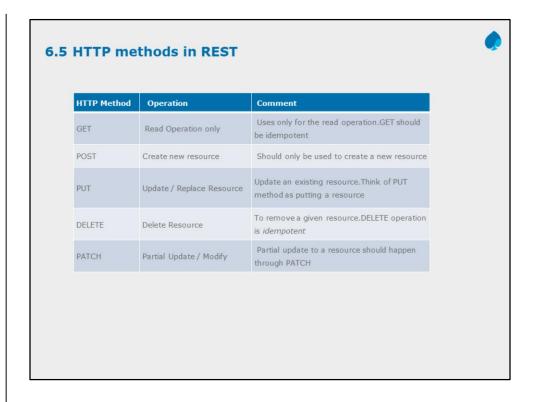
The Spring MVC uses a Controller component for handling the request. But a typical application may have several controllers. To determine which controller should handle the request, DispatcherServlet starts by querying one or more HandlerMappings. A HandlerMapping typically maps URL patterns to RestControllers.

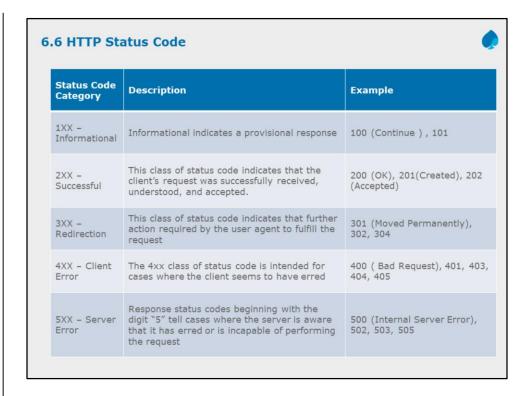
Once the DispatcherServlet has a appropriate RestController selected, it dispatches the request to that Controller which performs the business logic (a well-designed RestController object delegates responsibility of business logic to one or more service objects). Upon completion of business logic, HTTPResponse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

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.



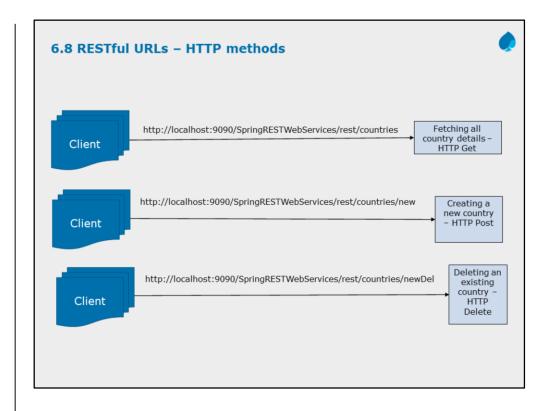


6.6 HTTP Status Code

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



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



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.

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

 $\label{local-problem} \mbox{http://localhost:9090/SpringRESTWebServices/rest/countries/3} \ \mbox{-> With this URL country details are fetched for country Id} = 3$

- 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:

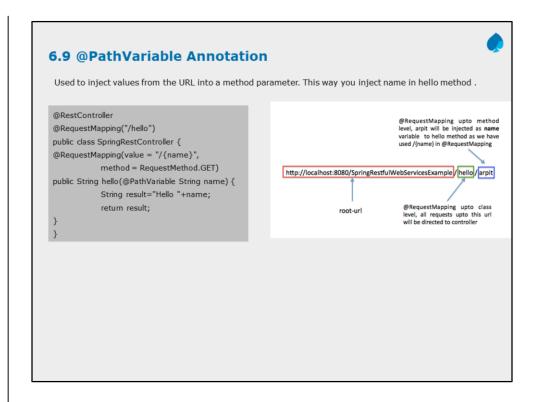
- <filter>
 - <filter-name>httpMethodFilter</filter-name>
- <filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>
- </filter>
- <filter-mapping>
 <filter-name>httpMethodFilter</filter-name>
- <servlet-name>dispatcher</servlet-name>
- </filter-mapping>

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

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

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

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.

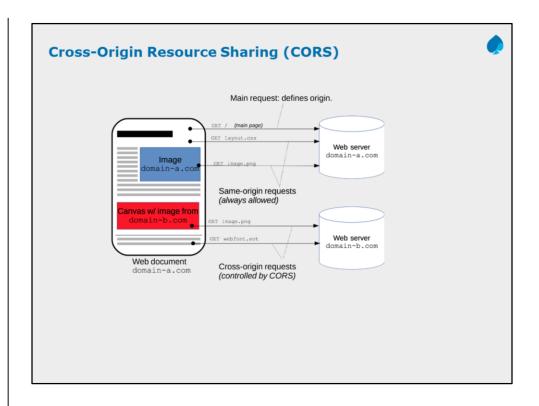
Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

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).

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



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.

```
@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()

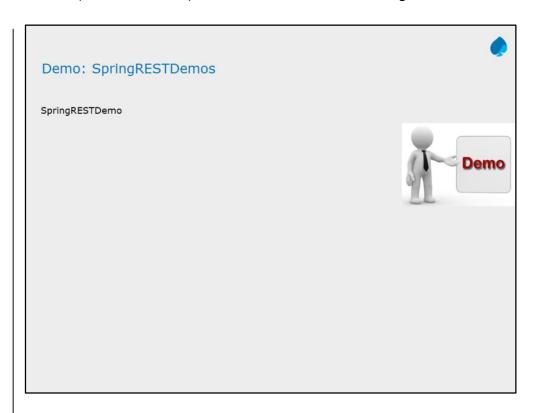
Head:

headForHeaders(String url, String... urlVariables)

Options:

optionsForAllow(String url, String... urlVariables)

These demos can be executed for better understanding



Summery

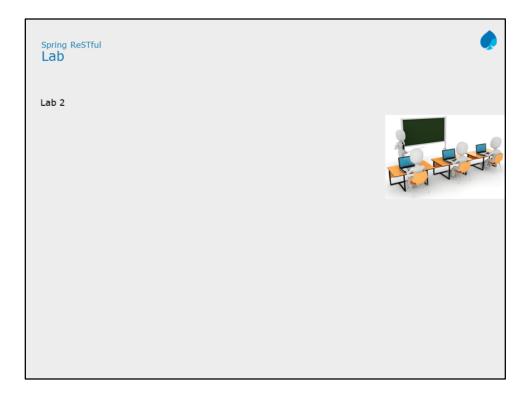
- Spring MVC REST Workflow
- SpringREST Introduction
- Life cycle of a Request in Spring MVC Restful
- Why REST Controller?
- HTTP methods in REST
- HTTP Status Code
- HTTP request Mapping
- RESTful URLs HTTP methods
- @PathVariable, @RequestBody Annotation
- ResponseEntity Object
- Cross-Origin Resource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



References:

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

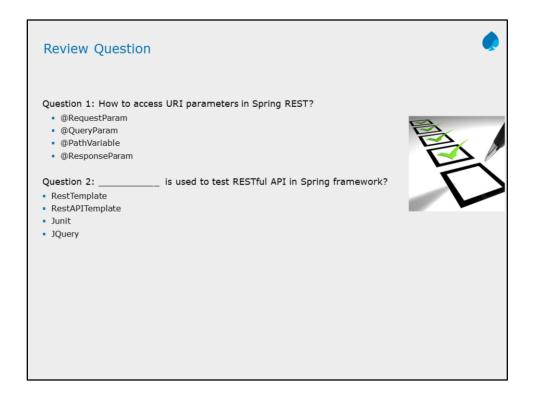
Corresponding lab assignment



Add the notes here.

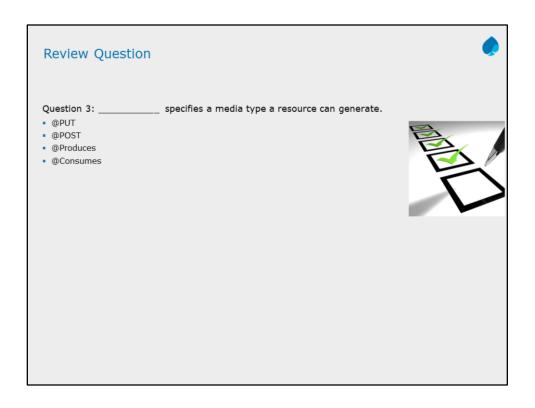
Question 1: Option 1, Option 2

Question 2: True



Add the notes here.

Question 3: @Produces



Add the notes here.