



Spring framework Core 5



Agenda

1. Presentación
2. Objetivos
3. Contenido
4. Despedida



3. Contenido

- i. Introducción a Spring Framework
- ii. Spring Core
- iii. Spring AOP
- iv. Spring JDBC – Transaction
- v. Spring ORM – Hibernate 5
- vi. Spring Data JPA
- vii. Fundamentos Spring MVC y Spring REST



vi. Fundamentos Spring MVC y Spring REST



vi. Fundamentos Spring MVC y Spring REST (a)

- i. ¿Qué es Spring MVC?
 - a. Patrón MVC
- ii. Dispatcher Servlet
 - a. Ciclo de vida de las peticiones HTTP
- iii. Configuración Spring MVC
 - a. WebApplicationContext
 - b. ContextLoaderListener
 - c. Namespace mvc
 - d. Sirviendo contenido estático



vi. Fundamentos Spring MVC y Spring REST (b)

iv. Controllers y Views

a. @Controller

b. @RequestMapping

c. @PathVariable

d. @RequestParam

e. InternalResourceViewResolver

Práctica 28. Hola Mundo Spring
MVC



vi. Fundamentos Spring MVC y Spring REST (c)

v. Formularios y redirección

a. Tags spring

b. Model Interface

c. @ModelAttribute

d. Forward y SendRedirect

e. @SessionAttributes

Práctica 29. Envío y recepción de formularios



vi. Fundamentos Spring MVC y Spring REST (d)

vi. Validaciones

a. Validator Interface

b. @InitBinder

Práctica 30 – Parte 1. Spring Validation

Práctica 30 – Parte 2. Validación de formularios con
Spring Validation



vi. Fundamentos Spring MVC y Spring REST (d)

vii. Spring REST

- a. ¿Qué es REST?
- b. Principios de REST
- c. @RequestBody y @ResponseBody
- d. @RestController
- e. Código de status HTTP

Práctica 31. Implementación Servicios REST



vi.i ¿Qué es Spring MVC?



Objetivos de la lección

vi.i ¿Qué es Spring MVC?

- Comprender el estilo arquitectónico Model-View-Controller (MVC).
- Comprender qué es Spring MVC
- Comprender las ventajas de utilizar Spring MVC.

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?



i. ¿Qué es Spring MVC?

a. Patrón MVC

vi.i ¿Qué es Spring MVC? (a)

- Spring MVC es la implementación del estilo arquitectónico model-view-controller de facto para Spring Framework para el desarrollo de aplicaciones web.
- Es ampliamente utilizado para desarrollar aplicaciones web, similar a JSF, Vaadin, Grails, Play 2 y Struts 1 y 2.
- Spring MVC, a diferencia de otros frameworks, permite la alta cohesión y el bajo acoplamiento en el desarrollo de aplicaciones web. Separa marcadamente la vista, el modelo y los controladores.

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?

vi.i ¿Qué es Spring MVC? (b)

- Ventajas de utilizar Spring MVC.
- Presenta una clara separación de roles o responsabilidades (controladores, validadores, objetos comando, formularios, modelos, etcétera).
- Configuración natural mediante IoC de Spring.
- Adaptabilidad, flexibilidad y no intrusión. Se vale de anotaciones para configurar el ambiente MVC.

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?

vi.i ¿Qué es Spring MVC? (c)

- Ventajas de utilizar Spring MVC.
- Reusabilidad de código de negocio (capa de servicio), total adaptabilidad mediante inyección de dependencias.
- Alta configuración de *binding (property editors)* y validación de objetos(formularios).
- Alta configuración de manejadores de URIs (*handler mappings*) y resolución de vistas (*view resolvers*).

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?

vi.i ¿Qué es Spring MVC? (d)

- Ventajas de utilizar Spring MVC.
- Flexible modelo de transferencia de objetos Modelo hacia la capa de presentación.
- Alta integración con diversos frameworks de presentación y plantillas.
- Inclusión de *Tag Libraries* propietaria de Spring MVC muy potente para el trabajo con formularios.
- Total integración con JSTL.

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?

vi.i ¿Qué es Spring MVC? (e)

- Ventajas de utilizar Spring MVC.
- Soporte Bean Validation JSR 303.
- Facilidad de implementación de Servicios REST mediante `@ResponseBody` o `@RestController`.
- Ampliamente utilizado para implementación de arquitecturas REST para múltiples dispositivos, así como alta integración con frameworks JavaScript (el futuro de las aplicaciones móviles).

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?



i. ¿Qué es Spring MVC?

a. Patrón MVC

vi.i ¿Qué es Spring MVC? (a)

- Patrón MVC.
- Spring MVC implementa el patrón (estilo arquitectónico) MVC el cual separa marcadamente los 3 aspectos principales de una aplicación web Modelo, Vistas y Controladores.
- El **modelo** encapsula las estructuras de datos (POJOs) de la aplicación en general, también conocida como *dominio*. También el modelo encapsula los objetos de acceso a datos (DAOs) que son los encargados de obtener la información de bases de datos.

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?

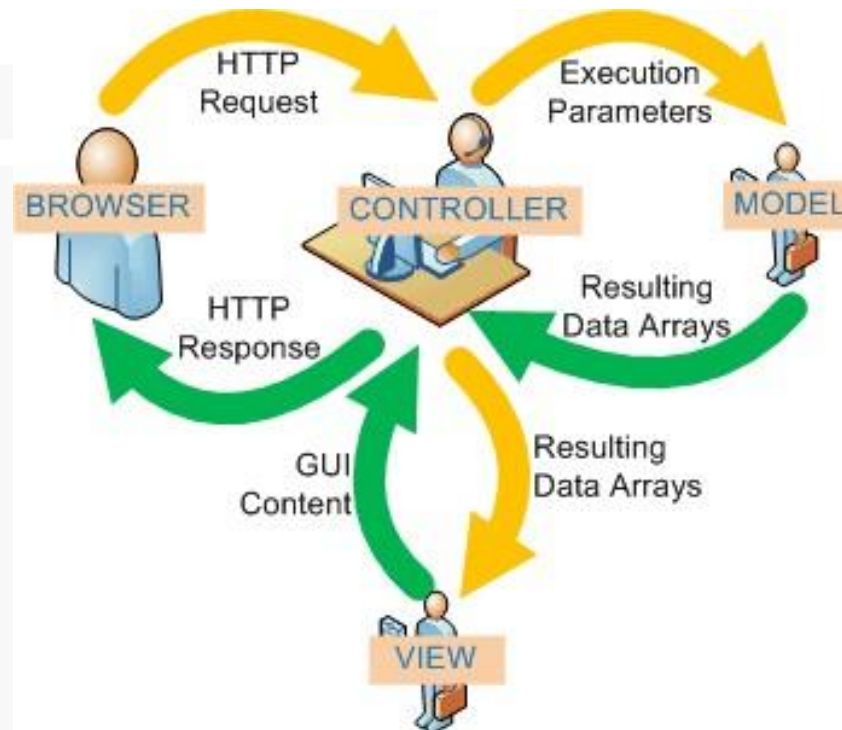
vi.i ¿Qué es Spring MVC? (b)

- Patrón MVC.
- La **vista**, también conocida como capa de presentación, es la responsable de mostrar la información correspondiente (modelos) al usuario. En ambiente web, la vista son documentos HTML que presentan información al usuario mediante un navegador web.
- El **controlador** es el responsable de procesar las peticiones que el usuario realiza. En ambiente web el controlador se encarga de atender peticiones HTTP mediante la llamada a los modelos y vistas correspondientes requeridos para atender la petición.

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?

vi.i ¿Qué es Spring MVC? (c)

- Patrón MVC.



vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?



Resumen de la lección

vi.i ¿Qué es Spring MVC?

- Comprendimos el patrón (estilo arquitectónico) MVC.
- Comprendimos qué es Spring MVC
- Comprendimos cuales son las ventajas de utilizar Spring MVC frente a otros frameworks.

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?



Esta página fue intencionalmente dejada en blanco.

vi. Fundamentos Spring MVC y Spring REST - vi.i ¿Qué es Spring MVC?



vi.ii Dispatcher Servlet



Objetivos de la lección

vi.ii Dispatcher Servlet

- Revisar el funcionamiento general de Spring MVC.
- Comprender el patrón de diseño *Front Controller*.
- Conocer el ciclo de vida de las peticiones HTTP en Spring MVC.

vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet



ii. **Dispatcher Servlet**

a. Ciclo de vida de las peticiones HTTP



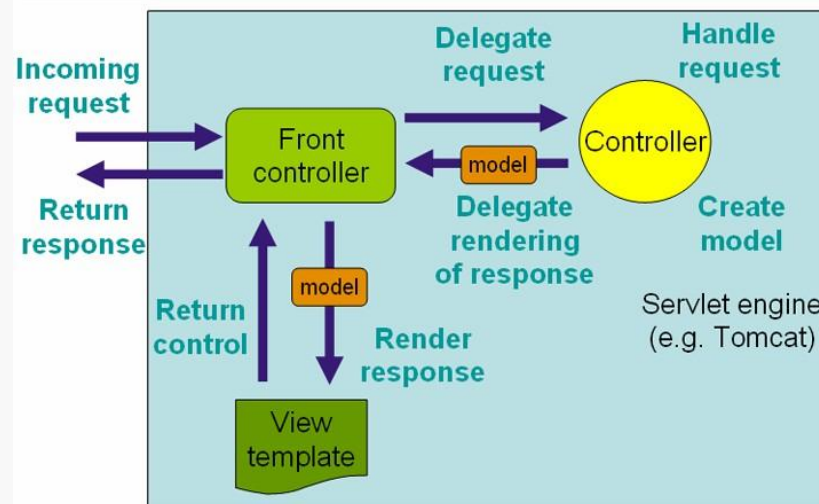
vi.ii Dispatcher Servlet (a)

- Spring MVC implementa el patrón de diseño J2EE *Front Controller*.
- El principal mecanismo de comunicación entre el cliente y el servidor en una aplicación Java web es el paradigma petición-respuesta.
- Spring MVC implementa el paradigma petición-respuesta mediante un único Servlet, el **DispatcherServlet**.

vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

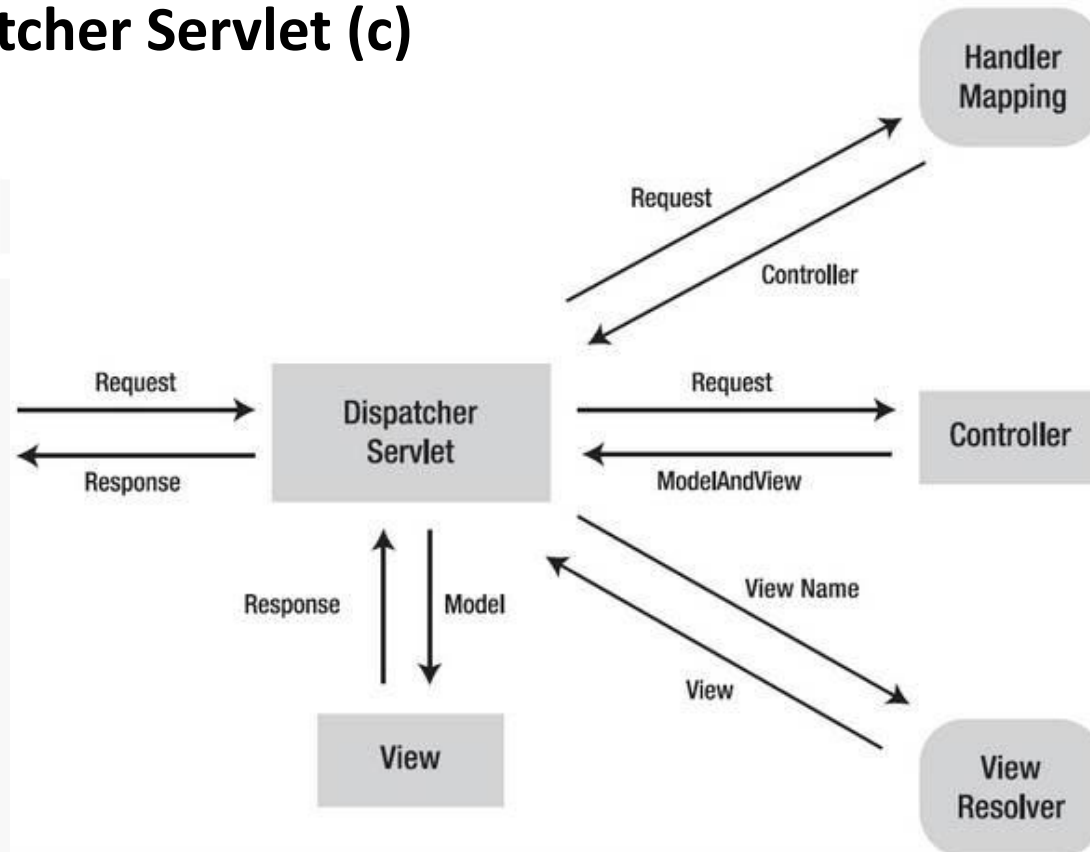
vi.ii Dispatcher Servlet (b)

- El **DispatcherServlet** es el *Front Controller* encargado de recibir todas las peticiones HTTP (Requests), así como del envío de las respuestas HTTP (Responses) correspondientes.



vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

vi.ii Dispatcher Servlet (c)



vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

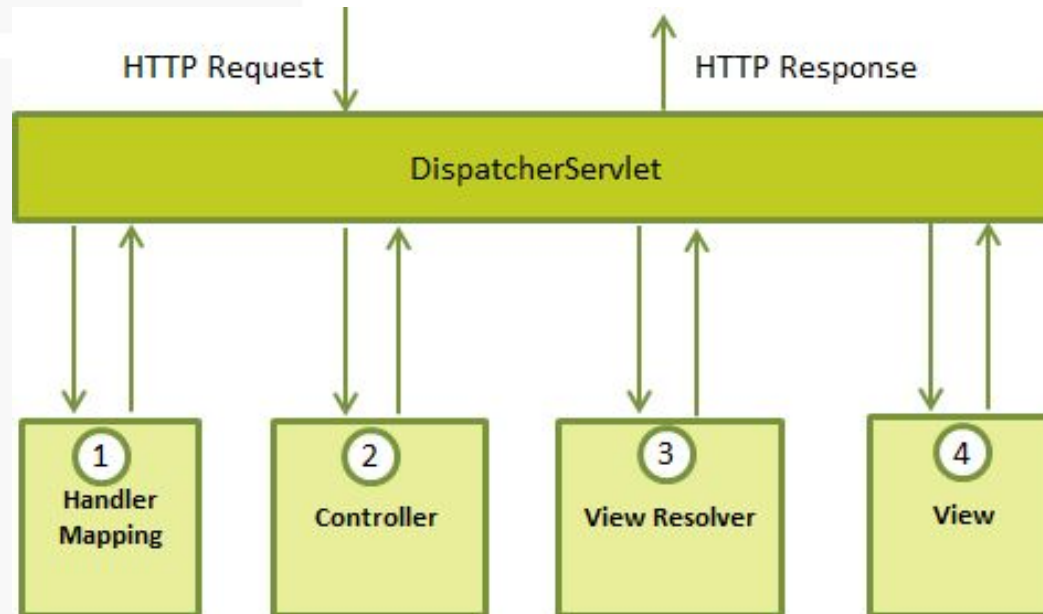


ii. Dispatcher Servlet

a. Ciclo de vida de las peticiones HTTP

vi.ii Dispatcher Servlet (a)

- Ciclo de vida de peticiones HTTP.

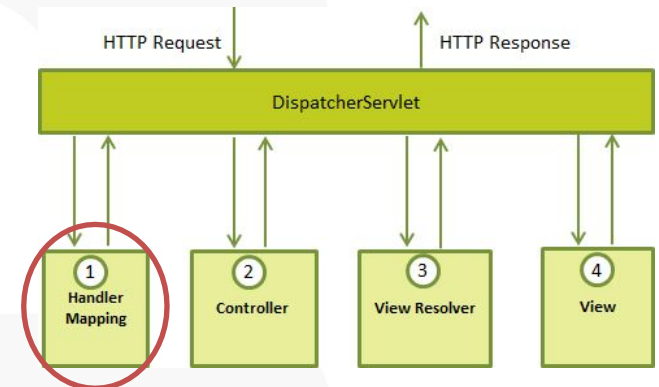


vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

vi.ii Dispatcher Servlet (b)

- Ciclo de vida de peticiones HTTP.

1. Después de recibir la petición, el **DispatcherServlet** consulta el *HandlerMapping* para conocer a que controlador delegar el procesamiento de la petición.



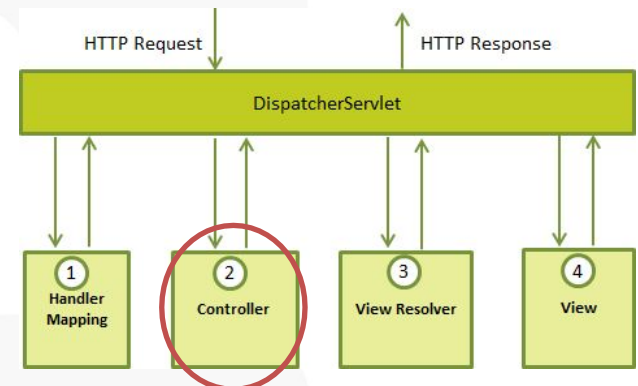
vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

vi.ii Dispatcher Servlet (c)

- Ciclo de vida de peticiones HTTP.

2. La petición es delegada al *controller* apropiado para su procesamiento. El controlador es un Bean de Spring y éste puede tener inyectadas dependencias para el procesamiento de la petición mediante llamadas a la capa de servicio.

El controlador retorna el modelo y la vista que debe mostrarse al cliente.



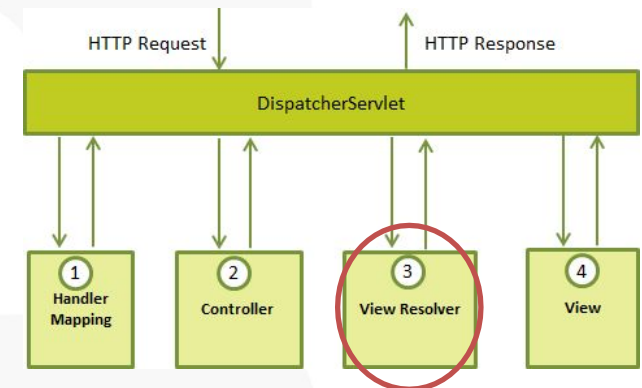
vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

vi.ii Dispatcher Servlet (d)

- Ciclo de vida de peticiones HTTP.

3. El **DispatcherServlet** delega la búsqueda de la vista correspondiente al *ViewResolver* configurado.

El ViewResolver devuelve la vista requerida.

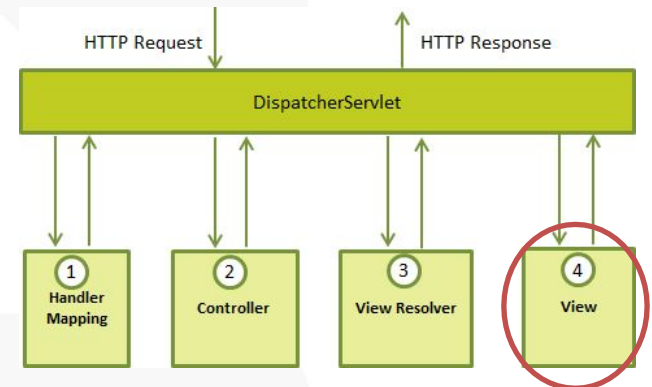


vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

vi.ii Dispatcher Servlet (e)

- Ciclo de vida de peticiones HTTP.

4. El **DispatcherServlet** envía la información (modelo) a presentar a la vista para que ésta sea enviada a través de la respuesta HTTP y por ende renegreada al usuario mediante un navegador web.



vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

vi.ii Dispatcher Servlet (f)

- Ciclo de vida de peticiones HTTP.
- El HandlerMapping, Controllers, y ViewResolvers son parte del **WebApplicationContext**, extensión de **ApplicationContext** para poder atender los requerimientos del paradigma petición-respuesta en aplicaciones Java web manejadas por Spring MVC.

vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet

Resumen de la lección

vi.ii Dispatcher Servlet

- Comprendimos el funcionamiento general de Spring MVC.
- Comprendimos como se implementa el patrón de diseño *Front Controller* en Spring MVC.
- Conocimos el Servlet base de Spring MVC que gestiona las peticiones HTTP.
- Revisamos a grandes rasgos los componentes *HandlerMapping*, *Controller*, *View* y *ViewResolver*.
- Analizamos el ciclo de vida de las peticiones HTTP en Spring MVC.

vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet



Esta página fue intencionalmente dejada en blanco.

vi. Fundamentos Spring MVC y Spring REST - vi.ii Dispatcher Servlet



vi.iii Configuración Spring MVC

Objetivos de la lección

vi.iii Configuración Spring MVC (a)

- Analizar los requerimientos mínimos para implementar aplicaciones web con Spring MVC.
- Revisar el contexto `WebApplicationContext`.
- Comprender la diferencia entre `RootApplicationContext` y `WebApplicationContext`.
- Verificar como se configura el `WebApplicationContext`.
- Comprender como se configura el `DispatcherServlet`.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

Objetivos de la lección

vi.iii Configuración Spring MVC (b)

- Verificar como configurar el RootApplicationContext.
- Analizar la utilidad del namespace mvc.
- Habilitar la configuración de Spring MVC mediante anotaciones.
- Analizar el comportamiento que deben tener los recursos estáticos tal como imágenes, archivos css y JavaScript.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC



iii. **Configuración Spring MVC**

- a. WebApplicationContext
- b. ContextLoaderListener
- c. Namespace mvc
- d. Sirviendo contenido estático

vi.iii Configuración Spring MVC (a)

- Es necesario considerar lo siguiente para configurar un proyecto maven con Spring MVC satisfactoriamente.
- Requerimientos:
 - Proyecto Maven war (packaging war)
 - Dependencias Maven requeridas
 - Descriptor de despliegue web.xml (opcional para API Servlet 3.0+)

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (b)

- Proyecto Maven war (packaging war)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
```

```
...>
```

```
<modelVersion>4.0.0</modelVersion>
```

```
<groupId>org.certificatic</groupId>
```

```
<artifactId>5-spring-web-mvc</artifactId>
```

```
<version>0.0.1-SNAPSHOT</version>
```

```
<packaging>war</packaging>
```

```
<dependencies>...</dependencies>
```

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

```
</project>
```

vi.iii Configuración Spring MVC (c)

- Dependencias Maven requeridas

```
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-web</artifactId>  
  <version>${org.springframework.version}</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.springframework</groupId>  
  <artifactId>spring-webmvc</artifactId>  
  <version>${org.springframework.version}</version>  
</dependency>
```

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (d)

- Descriptor de despliegue web.xml (opcional para API Servlet 3.0+)

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
    http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"  
  version="3.0">  
  
  <display-name>5-spring-web-mvc</display-name>  
  
  ...  
  
</web-app>
```

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC



iii. Configuración Spring MVC

- a. **WebApplicationContext**
- b. ContextLoaderListener
- c. Namespace mvc
- d. Sirviendo contenido estático

vi.iii Configuración Spring MVC (a)

- `WebApplicationContext`
- Es la implementación `ApplicationContext` de Spring Framework encargada de dar soporte para el despliegue de aplicaciones web utilizando Spring Web o Spring MVC.
- Una aplicación web al menos tiene un **`WebApplicationContext`** y éste está asociado a un único **`DispatcherServlet`**.
- Es necesario configurar el **`DispatcherServlet`** de Spring en el `web.xml` asignándole un nombre identificativo al Servlet.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (b)

- `WebApplicationContext`
- La configuración por defecto de un **WebApplicationContext** asociado a un **DispatcherServlet** será definida por el Bean Configuration File `<servlet-name>-servlet.xml`, donde `<servlet-name>` es el nombre del **DispatcherServlet**.
- En una aplicación web es posible configurar múltiples **DispatcherServlet** que manejen distintas peticiones HTTP, por ejemplo distintos paths (`http://localhost:8080/alumnos/*`, `http://localhost:8080/maestros/*`) también conocidos como *RequestMappings*.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (c)

- `WebApplicationContext`
- Toda aplicación web, con múltiples **WebApplicationContext** pueden compartir un único **RootApplicationContext** (`ApplicationContext`) la cual define beans de tipo:
 - Capa de servicio (*service-layer*)
 - Capa de acceso a datos (*dao-layer, repositories*)
 - Componentes de utilería (*utils*)
 - Aspectos (*spring-aop, aspectj*)
 - Beans de infraestructura
 - Entre otros beans componentes

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

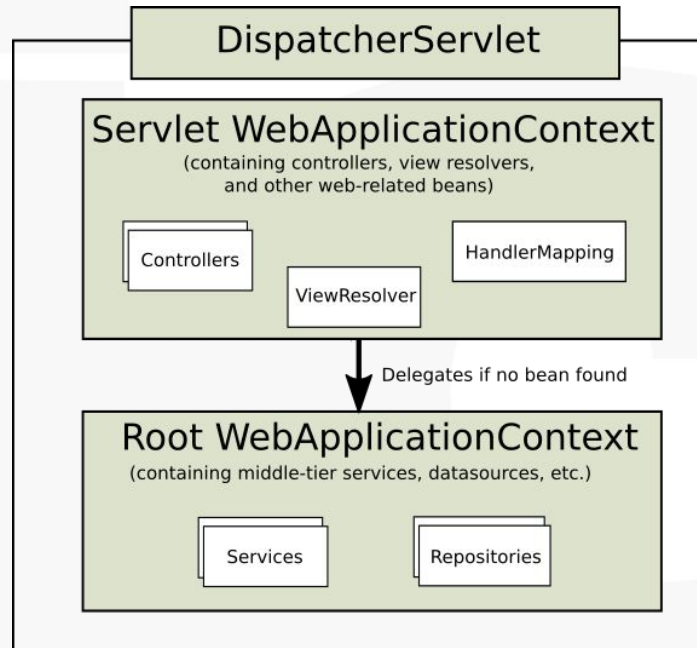
vi.iii Configuración Spring MVC (d)

- `WebApplicationContext`
- El `RootApplicationContext` es accesible para todos los `WebApplicationContext` configurados (lo hereda), sin embargo el `RootApplicationContext` no puede acceder a los `WebApplicationContext`.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (e)

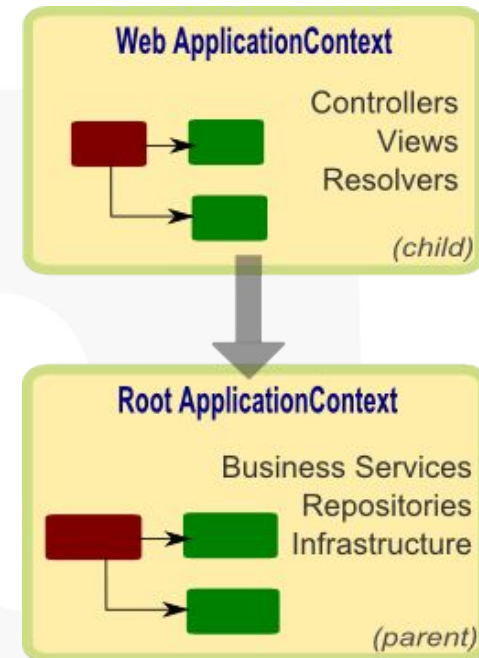
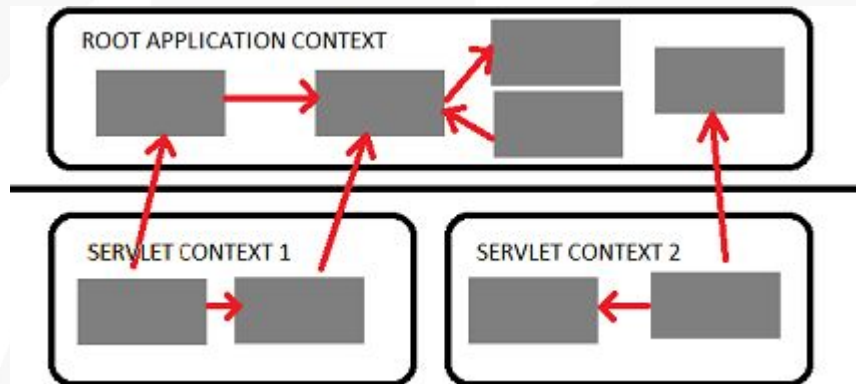
- WebApplicationContext



vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (f)

- WebApplicationContext



vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (g)

- WebApplicationContext: Configuración web.xml (a)

```
<!-- DispatcherServlet (Spring Front Controller) -->
<servlet>
  <servlet-name>dispatcherServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>

<!-- RequestMapping dispatcherServlet Servlet -->
<servlet-mapping>
  <servlet-name>dispatcherServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Por defecto, Spring tratará de buscar el archivo **dispatcherServlet-servlet.xml** en **webapp/WEB-INF/**

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (h)

- WebApplicationContext: Configuración web.xml (b)

```
<!-- DispatcherServlet (Spring Front Controller) -->
<servlet>
  <servlet-name>dispatcherServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/mvc/dispatcherServlet-servlet.xml</param-value>
  </init-param>
</servlet>

<!-- RequestMapping dispatcherServlet Servlet -->
<servlet-mapping>...</servlet-mapping>
```

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC



iii. Configuración Spring MVC

- a. WebApplicationContext
- b. ContextLoaderListener
- c. Namespace mvc
- d. Sirviendo contenido estático

vi.iii Configuración Spring MVC (a)

- ContextLoaderListener
- Se encarga de levantar el **RootApplicationContext** de la aplicación web Spring MVC.
- Se configura mediante un **ServletContextListener**, que se dispara cada que el contexto web se inicializa o se destruye. (Existen muchos tipos de Lsiteners documentados en el API Servlet).

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (b)

- ContextLoaderListener
- El listener **ContextLoaderListener** utiliza **<context-param>** (definido en web.xml) para definir el parámetro **contextConfigLocation** requerido para ubicar el archivo contenedor de la configuración de beans XML del **RootApplicationContext**.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (c)

- ContextLoaderListener: Configuración web.xml

```
<!-- RootApplicationContext contextConfigLocation -->
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring/root-application-context.xml</param-value>
</context-param>

<!-- RootApplicationContext ContextLoaderListener -->
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC



iii. Configuración Spring MVC

- a. WebApplicationContext
- b. ContextLoaderListener
- c. Namespace mvc
- d. Sirviendo contenido estático

vi.iii Configuración Spring MVC (a)

- Namespace mvc
- Por defecto la implementación MVC que provee Spring está deshabilitada, para ello es necesario habilitar el namespace mvc.



vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (b)

- Namespace mvc

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation=" http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">
  ...
</beans>
```

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (c)

- Namespace mvc
- Habilita configuración de beans por defecto así como atajos de configuración de beans mediante el tag <mvc> para:
 - HandlerMapping
 - HandlerAdapter
 - HandlerExceptionResolver
 - ViewResolver
 - Entre otros

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (d)

- Namespace mvc
- Habilitar configuración de controladores Spring MVC por medio de @Anotaciones:

```
<mvc:annotation-driven/>
```

ó

```
@EnableWebMvc
```

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (e)

- Namespace mvc
- `<context:component-scan />` no habilita la configuración de controladores de Spring MVC, es necesario habilitar la configuración por anotaciones con `<mvc:annotation-driven />`
- Se habilitan las anotaciones `@RequestMapping`, `@PathVariable`, `@ModelAttribute`, `@RequestParam`, entre otros.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC



iii. Configuración Spring MVC

- a. WebApplicationContext
- b. ContextLoaderListener
- c. Namespace mvc
- d. Sirviendo contenido estático

vi.iii Configuración Spring MVC (a)

- Sirviendo contenido estático.
- La configuración del **DispatcherServlet** mediante el tag `<url-pattern>/</url-pattern>` recibirá todas las peticiones entrantes a la aplicación web que cumplan con el patrón “/”.
- Las peticiones de recursos estáticos como imágenes, documentos css o js, no requieren pasar a lo largo del ciclo de vida del Request manejado por el **DispatcherServlet**.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

vi.iii Configuración Spring MVC (b)

- Sirviendo contenido estático.
- Es posible habilitar un mapeo de recursos para la aplicación que omita el ciclo de vida del Request en una aplicación Spring MVC mediante:

```
<mvc:resources mapping="/resources/**" location="/static/theme1/"  
cache-period="31556926"/>
```

- Se sugiere habilitar un tiempo de cacheo para los recursos estáticos.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC

Resumen de la lección

vi.iii Configuración Spring MVC

- Comprendimos como las diferencias entre `WebApplicationContext` y `RootApplicationContext`.
- Verificamos como habilitar Spring MVC a través de su configuración en el archivo de despliegue `web.xml`
- Comprendimos cuales son los requerimientos (dependencias) necesarias para implementar aplicaciones web con Spring MVC.
- Habilitamos la configuración de Controladores mediante anotaciones.
- Comprendimos la utilidad del namespace `mvc`.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC



Esta página fue intencionalmente dejada en blanco.

vi. Fundamentos Spring MVC y Spring REST - vi.iii Configuración Spring MVC



vi.iv Controllers y Views



Objetivos de la lección

vi.iv Controllers y Views

- Revisar las principales anotaciones de Spring MVC tal como `@Controller`, `@RequestMapping`, `@PathVariable` y `@RequestParam`.
- Comprender como se aplican éstas distintas anotaciones.
- Comprender como se relaciona el *HandlerMapping* con `@RequestMapping`.
- Comprender que es un Controlador.
- Analizar para qué sirve un `ViewResolver`.
- Implementar un ejemplo Hola Mundo de Spring MVC.

vi. Fundamentos Spring MVC y Spring REST - vi.iv Controllers y Views



iv. Controllers y Views

- a. @Controller
 - b. @RequestMapping
 - c. @PathVariable
 - d. @RequestParam
 - e. InternalResourceViewResolver
- Práctica 28. Hola Mundo Spring
MVC

vi.iv Controllers y Views (a)

- @Controller
- El **DispatcherServlet** delega el procesamiento de la petición HTTP al controlador, en específico a un método *handler method* en el controlador.
- La anotación **@Controller** (estereotipo de @Component) crea como bean la clase controladora anotada y habilita el uso de otras anotaciones relacionadas con los controladores de Spring MVC.

vi.iv Controllers y Views (b)

- @Controller
- Dado que una clase **@Controller** es un bean de Spring, es posible crear inyección de dependencias, preferentemente de abstracciones de capa de servicio (*service-layer, business-layer*).

@Controller

```
public class AccountManagementCustomerController {
```

@Autowired

```
private IAccountService accountService;
```

@Autowired

```
private IMovementService movementService;
```

```
}
```

vi.iv Controllers y Views (c)

- @Controller
- Los métodos de los controladores que manejan las peticiones HTTP, *handler methods*, deben devolver una Vista, una Vista y un Modelo o implementar una forma para responder al cliente un *request* válido.
- Por lo regular los métodos *handler methods* devuelven el nombre de la vista a responder (String).



iv. Controllers y Views

- a. @Controller
 - b. @RequestMapping
 - c. @PathVariable
 - d. @RequestParam
 - e. InternalResourceViewResolver
- Práctica 28. Hola Mundo Spring
MVC

vi.iv Controllers y Views (a)

- @RequestMapping
- Para que el **DispatcherServlet** sepa a que clase controladora delegar (y a que método de la misma), la anotación **@RequestMapping** es utilizada para poder mapear una URL a un método de una clase controladora anotada con **@Controller**.
- Los métodos que el controlador implementa con la anotación **@RequestMapping** se conocen como *handler methods*.



vi.iv Controllers y Views (b)

- @RequestMapping
- La anotación **@RequestMapping** es aplicable a tipos (clases) y métodos.
- Cuando se aplica a una clase, el **@RequestMapping** definido se hereda a todos sus demás *handler methods*.

vi.iv Controllers y Views (c)

- @RequestMapping

@Controller

```
@RequestMapping(value =  
"/customer/manage/accounts") public class  
AccountManagementCustomerController {
```

```
    @RequestMapping(value = "/view", method =  
    RequestMethod.GET)  
    public String showViewAccountPage(Model model) {  
    }  
    ...  
    return "view_accounts";
```

El método **showViewAccountPage** mapea peticiones
HTTP GET a la URL:
/customer/manage/accounts/view

vi.iv Controllers y Views (d)

- @RequestMapping y @Controller

@Controller

@RequestMapping(value =

"/customer/manage/accounts") public class

AccountManagementCustomerController {

@RequestMapping(value = "/view", method =

RequestMethod.GET) public String **showViewAccountPage**(Model
model) {

...

} **return "view_accounts";**

}

pueden devolver múltiples tipos sin embargo, todos
Es importante destacar que los *handler methods*
ellos devuelven una **Vista** o una **Vista y un Modelo**.

vi.iv Controllers y Views (e)

- @RequestMapping
- La anotación **@RequestMapping** por defecto atiende cualquier método HTTP, ya sea GET, POST, PUT, DELETE, PATCH, entre otros.
- Es necesario especificar, en el atributo *method*, que métodos HTTP pueden ser atendidos por el *handler method* anotado con **@RequestMapping**.

```
@RequestMapping(value = { "/", "" }, method = RequestMethod.GET)
```



iv. Controllers y Views

- a. @Controller
 - b. @RequestMapping
 - c. @PathVariable
 - d. @RequestParam
 - e. InternalResourceViewResolver
- Práctica 28. Hola Mundo Spring
MVC

vi.iv Controllers y Views (a)

- `@PathVariable`
- La anotación **`@PathVariable`**, utilizada en conjunto con **`@RequestMapping`**, se utiliza para poder obtener una parte de la URL como parámetro, mismo que viene implícito en la URL al estilo de REST.
- Para obtener un parámetro **`@PathVariable`**, primero es necesario definir un *URI Template* en el valor del **`@RequestMapping`**.
- **`@PathVariable`** es aplicable a los parámetros del *handler method*.

vi.iv Controllers y Views (b)

- @PathVariable
- Definición de *URI Template* en el valor del **@RequestMapping**.

```
@RequestMapping(value = "/view/{accountId}", method = RequestMethod.GET)
public String showViewAccountPage(Model model, @PathVariable Long accountId) {
    ...
    return "view_accounts";
}
```

- La petición **GET** HTTP */customer/manage/accounts/view/123123* será manejada por el *handler method* showViewAccountPage.

vi.iv Controllers y Views (c)

- @PathVariable
- **@PathVariable** convierte en automático el valor del *URI Template* al tipo especificado del parámetro anotado (*binding*).

```
@RequestMapping(value = "/view/{accountId}", method = RequestMethod.GET)
public String showViewAccountPage(Model model, @PathVariable Long
accountId){...}
```

vi.iv Controllers y Views (d)

- @PathVariable
- El *binding* entre el *URI Template* y **@PathVariable** se realiza en automático siempre y cuando el nombre del *URI Template* y del parámetro anotado coincida. De no ser así, es posible indicar el nombre del *URI Template* a *bindear* en la anotación **@PathVariable**.

```
@RequestMapping(value = "/view/{accountId}", method =  
RequestMethod.GET) public String showViewAccountPage(Model model,  
                                                         @PathVariable("accountId") Long  
                                                         id){...}
```



iv. **Controllers y Views**

- a. @Controller
 - b. @RequestMapping
 - c. @PathVariable
 - d. @RequestParam
 - e. InternalResourceViewResolver
- Práctica 28. Hola Mundo Spring
MVC

vi.iv Controllers y Views (a)

- @RequestParam
- La anotación **@RequestParam** se utiliza para poder obtener parámetros de la petición (*request parameters*) mismos que viajan al final de la URL mediante la forma:
`<some-url>?param1=value1¶m2=value2&...¶mN=valueN`
- Para obtener un *request parameter* mediante **@RequestParam**, es necesario anotar el parámetro del *handler method* donde se requiera obtener el *request parameter*.

vi.iv Controllers y Views (b)

- @RequestParam
- El *binding* entre el *request parameter* (que viaja en la URL) y **@RequestParam** se realiza en automático siempre y cuando el nombre del *request parameter* y del parámetro anotado coincida.

```
@RequestMapping(value = "/view", method =  
RequestMethod.GET) public String showViewAccountPage(Model  
model,
```

```
@RequestParam Long id){...}
```

- La petición **GET** HTTP **/customer/manage/accounts/view?id=123123** será manejada por el *handler method* showViewAccountPage.

vi.iv Controllers y Views (c)

- @RequestParam
- De no coincidir el nombre del *request parameter* y del parámetro anotado con **@RequestParam**, es necesario indicar el nombre del *request parameter* en la anotación para que pueda realizarse el *binding*.

```
@RequestMapping(value = "/view", method =  
RequestMethod.GET) public String showViewAccountPage(Model  
model,
```

```
@RequestParam("id") Long accountId){...}
```

- La petición **GET** HTTP **/customer/manage/accounts/view?id=123123** será manejada por el *handler method* showViewAccountPage.

vi. Fundamentos Spring MVC y Spring REST - vi.iv Controllers y Views



iv. **Controllers y Views**

- a. @Controller
- b. @RequestMapping
- c. @PathVariable
- d. @RequestParam

e. InternalResourceViewResolver

Práctica 28. Hola Mundo Spring
MVC

vi.iv Controllers y Views (a)

- InternalResourceViewResolver
- La interface **ViewResolver** permite asignar y mapear un nombre a una vista con la vista misma mediante un objeto **View**, lo cual permite que el **ViewResolver** se desacople de la tecnología de presentación a implementar tal como JSPs, Velocity, XSLT, Tiles, Thymeleaf, entre otros.
- La interface View prepara el *request*, para asignar los modelos en la vista, y maneja el *response* que responderá el **DispatcherServlet** mismo que será *rendereado* al cliente (navegador web).

vi.iv Controllers y Views (b)

- InternalResourceViewResolver
- Existen distintas implementaciones **ViewResolver** implementadas por Spring, así como por terceros que desean integración de sus plantillas para la implementación de aplicaciones web con Spring MVC. Tal es el caso de Thymeleaf con su implementación:

org.thymeleaf.spring4.view.ThymeleafViewResolver

vi.iv Controllers y Views (c)

- InternalResourceViewResolver
- Como buena práctica y para respetar el patrón MVC, se recomienda que las vistas (JSPs) no sean accesibles desde el cliente, por tanto las vistas deben incluirse en la carpeta “WEB-INF”.
- Las vistas que se encuentra en “WEB-INF” son llamadas, *internal resource views*, por tanto Spring MVC implementa el **ViewResolver InternalResourceViewResolver** para poder acceder a éstas vistas.

vi.iv Controllers y Views (d)

- InternalResourceViewResolver
- Configuración básica ViewResolver.

```
<bean id="viewResolver"  
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
  <property name="viewClass"  
            value="org.springframework.web.servlet.view.JstlView" />  
  <property name="prefix" value="/WEB-INF/views/" />  
  <property name="suffix" value=".jsp" />  
</bean>
```


vi.iv Controllers y Views (e)

- InternalResourceViewResolver: Ejemplo

```
@RequestMapping(value = "/view/{accountId}", method = RequestMethod.GET)
public String showViewAccountPage(Model model, @PathVariable Long accountId) {
    ...
    return "view_accounts";
}
```

- Mediante la configuración de **InternalResourceViewResolver** anterior, Spring MVC responderá, a través del *response*, el contenido de la vista *view_accounts* ubicado en *"/WEB-INF/views/view_accounts.jsp"*.



iv. **Controllers y Views**

- a. @Controller
- b. @RequestMapping
- c. @PathVariable
- d. @RequestParam
- e. InternalResourceViewResolver

**Práctica 28. Hola Mundo Spring
MVC**

vi.iv Controllers y Views. Práctica 28. (a)

- Práctica 28. Hola Mundo Spring MVC
- Implementar desde cero la configuración necesaria para implementar una aplicación web con Spring MVC (proyecto 5-spring-web-mvc-hello-world).
- Comprender como se configura el `WebApplicationContext` y el `RootApplicationContext`.
- Configurar un bean `InternalResourceViewResolver`.
- Mostrar una vista básica mediante JSP imprimiendo un mensaje enviado desde el servidor.



Resumen de la lección

vi.iv Controllers y Views

- Revisamos para qué sirven y cómo se usan las principales anotaciones de Spring MVC tal como @Controller, @RequestMapping, @PathVariable y @RequestParam.
- Comprendimos como se delegan las peticiones a los *handler methods*.
- Analizamos como recibir parámetros en los *handler methods*.
- Comprendimos que es un ViewResolver.
- Configuramos una aplicación web simple con Spring MVC.



Esta página fue intencionalmente dejada en blanco.



vi.v Formularios y redirección

Objetivos de la lección

vi.v Formularios y redirección (a)

- Revisar a grandes rasgos algunos de los Tags de Spring para la implementación de formularios en la capa de presentación.
- Comprender como funciona el envío de formularios desde la vista al controlador mediante *binding*.
- Comprender la Interface Model y como se relaciona con el *binding* de formularios.
- Crear un formulario básico mismo que recuperaremos sus valores a través de objetos (*binding*) y no directamente del *request*.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



Objetivos de la lección

vi.v Formularios y redirección (b)

- Revisar la clase ModelAndView y en que casos nos es útil implementarla como retorno en los controladores.
- Comprender el uso de la anotación @ModelAttribute.
- Comprender la diferencia entre forward y sendRedirect en el API de Servlets.
- Analizar como funciona el mecanismo de sesiones en Spring MVC.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



v. Formularios y redirección

a. Tags Spring

b. Model Interface

c. @ModelAttribute

d. Forward y SendRedirect

e. @SessionAttributes

Práctica 29. Envío y recepción de formularios

vi.v Formularios y redirección (a)

- Tags Spring
- Para agilizar la creación de vistas basadas en JSP, Spring MVC implementa *Tag Libraries*, que conjunto con JSTL (*Java Server Pages Standard Tag Library*) apoya al tratamiento de variables provenientes del controlador sin necesidad de utilizar *snippets* y manipular el *request* directamente.
- Las *Tag Libraries* de Spring apoyan a la evaluación de mensajes de error, internacionalización, evaluación de expresiones con SpEL, escape de entidades HTML, entre otros.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (b)

- Tags Spring
- Existen muchos Tags de Spring que apoyan para la visualización de modelos en las páginas JSP, entre las más usadas están:
 - form
 - Input
 - checkbox
 - checkboxes
 - radiobutton
 - radiobuttons
 - password
 - select
 - option
 - options
 - textarea
 - entre otros

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (c)

- Tags Spring
- Principalmente existen 2 *Tag Libraries* core de Spring:
 - spring JSP tag Library:
`<%@ taglib uri="http://www.springframework.org/tags" prefix="spring"%>`
 - spring-form JSP Tag Library:
`<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>`
- Existen otras Tags de Spring Security, se analizarán más adelante.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (d)

- Tags Spring
- Para mayor referencia consultar:
 - <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/spring-tld.html>
 - <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/spring-form-tld.html>

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



v. Formularios y redirección

a. Tags Spring

b. **Model Interface**

c. @ModelAttribute

d. Forward y SendRedirect

e. @SessionAttributes

Práctica 29. Envío y recepción de formularios

vi.v Formularios y redirección (a)

- Model Interface
- La interface *Model* se utiliza para enviar datos, parámetros, información a las vistas.
- La clase *ModelMap* es la principal implementación de *Model*, utilizada para agregar parámetros y enviarlas a la vista.
- En términos técnicos, la interface *Model* se utiliza como adaptador para agregar los modelos al *request*.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (b)

- Model Interface: Ejemplo

```
@RequestMapping(value = "/customer/accounts", method =  
RequestMethod.GET) public String showManagementAccountsPage(Model  
model,
```

```
        @ModelAttribute("logableUser") Customer customer) {  
    List<Account> accounts =  
  
    accountService.getByCustomerId(customer.getId());
```

```
    model.addAttribute("accounts", accounts);
```

```
    return "view_accounts";  
}
```

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



vi.v Formularios y redirección (c)

- Model Interface
- Existen otras clases tal como *ModelAndView* que contiene como atributo un *Model*, el cual también es ampliamente utilizado para enviar modelos a la vista sin embargo, este objeto también encapsula el nombre de la vista a enviar.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (d)

- Model Interface: Ejemplo ModelAndView

```
@RequestMapping(value = "/customer/accounts", method =  
RequestMethod.GET) public ModelAndView showManagementAccountsPage(  
    @ModelAttribute("logableUser") Customer customer) {  
  
    List<Account> accounts = accountService.getByCustomerId(customer.getId());  
  
    ModelAndView mav = new ModelAndView("view_accounts");  
    model.addObject("accounts", accounts);  
  
    return mav;  
}
```

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



v. Formularios y redirección

- a. Tags Spring
- b. Model Interface
- c. **@ModelAttribute**
- d. Forward y SendRedirect
- e. @SessionAttributes

Práctica 29. Envío y recepción de formularios

vi.v Formularios y redirección (a)

- @ModelAttribute
- La anotación @ModelAttribute es ampliamente utilizada en muchos ambitos relativos a Spring MVC.
- Utilizando la anotación @ModelAttribute es posible enviar modelos a la vista de manera generalizada para todos los @RequestMapping del controlador en cuestión.
- Otra funcionalidad importante de @ModelAttribute es que permite obtener objetos almacenados en sesión (se revisará más adelante).

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (b)

- @ModelAttribute
- Principalmente utilizando la anotación @ModelAttribute es posible obtener los valores ingresados en un formulario directamente sobre un objeto POJO en los *handler methods*.
- @ModelAttribute se utiliza como mecanismo de *binding* entre un objeto modelo (POJO) en el controlador y un formulario o comando (*command*) en la vista y, viceversa.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

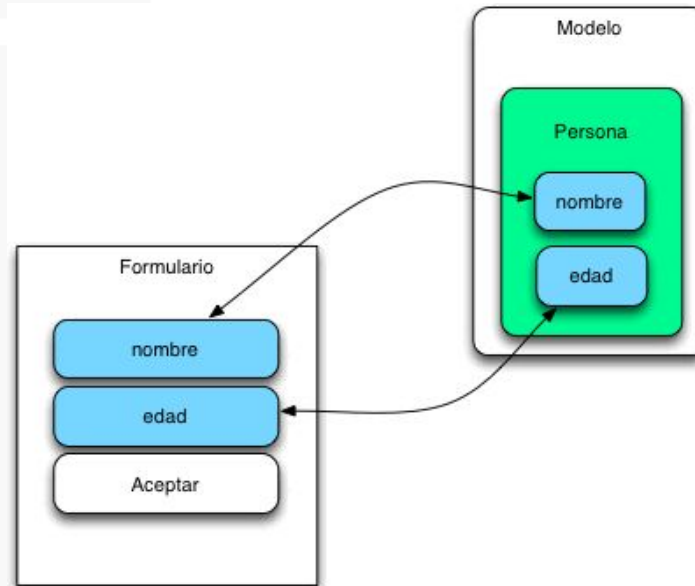
vi.v Formularios y redirección (c)

- @ModelAttribute
- ¿Cómo recibir datos desde un formulario en la vista?
 - Recibiendo el modelo o comando previamente enviado al formulario para poder obtener los valores del mismo.
- ¿Cómo envío el modelo o comando al formulario?
 - Mandando un objeto nuevo (new) al request mediante la interface Model.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (d)

- *Binding* de un modelo o comando mediante `@ModelAttribute`



vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (e)

- Envío el modelo o comando al formulario (a)

```
@RequestMapping(value = "/customer/create", method = RequestMethod.GET)
```

```
public String showCreateCustomerPage(Model model) {
```

```
    Customer customer = new Customer();
```

```
    model.addAttribute("createCustomerForm", customer);
```

```
    return
```

```
} "create_customer";
```

```
<form:form id="form" action="/customers/createCustomer"
           commandName="createCustomerForm" method="POST">
```

```
    Name: <form:input path="name" tabindex="1" /> <br />
```

```
    Last Name: <form:input path="lastName" tabindex="2" /> <br />
```

```
    <input class="button" type="submit" value="Create" />
</form:form>
```

/WEB-INF/views/create_customer.jsp

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (f)

- Envío el modelo o comando al formulario (b)

```
@RequestMapping(value = "/customer/create", method = RequestMethod.GET)
public String showCreateCustomerPage(@ModelAttribute Customer createCustomerForm) {
    return "create_customer";
}
```

```
<form:form id="form" action="/customers/createCustomer"
           commandName="createCustomerForm" method="POST">

    Name: <form:input path="name" tabindex="1" /> <br />
    Last Name: <form:input path="lastName" tabindex="2" /> <br />

    <form:input class="button" type="submit" value="Create" />
</form>
```

/WEB-INF/views/create_customer.jsp

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (g)

- @ModelAttribute sobre parámetros de un *handler method*.
- Cuando anotamos un parámetro de un *handler method* con la @ModelAttribute significa una de dos cosas:
 - El objeto anotado viene en el modelo (*request o session*) y éste debe recuperarse.
 - Si el objeto anotado no existe en el modelo, el objeto debe ser instanciado y agregado al modelo.

```
@RequestMapping(value = "/customer/create", method = RequestMethod.GET)
public String showCreateCustomerPage(@ModelAttribute Customer createCustomerForm) {
    return "create_customer";
}
```

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (h)

- Recepción de modelo o comando desde un formulario

```
<form:form id="form" action="/customers/createCustomer" commandName="createCustomerForm"
method="POST">
```

```
Name: <form:input path="name" tabindex="1" /> <br />
```

```
Last Name: <form:input path="lastName" tabindex="2" /> <br />
```

```
<input class="button" type="submit" value="Create" />
```

```
</form:form>
```

/WEB-INF/views/create_customer.jsp

```
@RequestMapping(value = "/customers/createCustomer",
method = RequestMethod.POST)
public String createCustomer(@ModelAttribute("createCustomerForm")
Customer createCustomerForm) {
    customerService.create(createCustomerForm);
    return "redirect:/customers/viewAll";
}
```

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (i)

- Recepción de modelo o comando desde un formulario
- Es posible utilizar el atributo **modelAttribute** en lugar de **commandName** en la etiqueta <form:form />

```
<form:form id="form" action="/customers/createCustomer" commandName="createCustomerForm"
method="POST">
```

ó

```
<form:form id="form" action="/customers/createCustomer" modelAttribute="createCustomerForm"
method="POST">
```

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



v. Formularios y redirección

- a. Tags Spring
- b. Model Interface
- c. @ModelAttribute
- d. Forward y sendRedirect
- e. @SessionAttributes

Práctica 29. Envío y recepción de formularios

vi.v Formularios y redirección (a)

- Forward y sendRedirect
- Los métodos forward y sendRedirect son las dos estrategias que proporciona el API Servlet para redirigir el flujo de una petición HTTP (*request*) hacia una vista, otro servlet (controlador) u otro recurso.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (b)

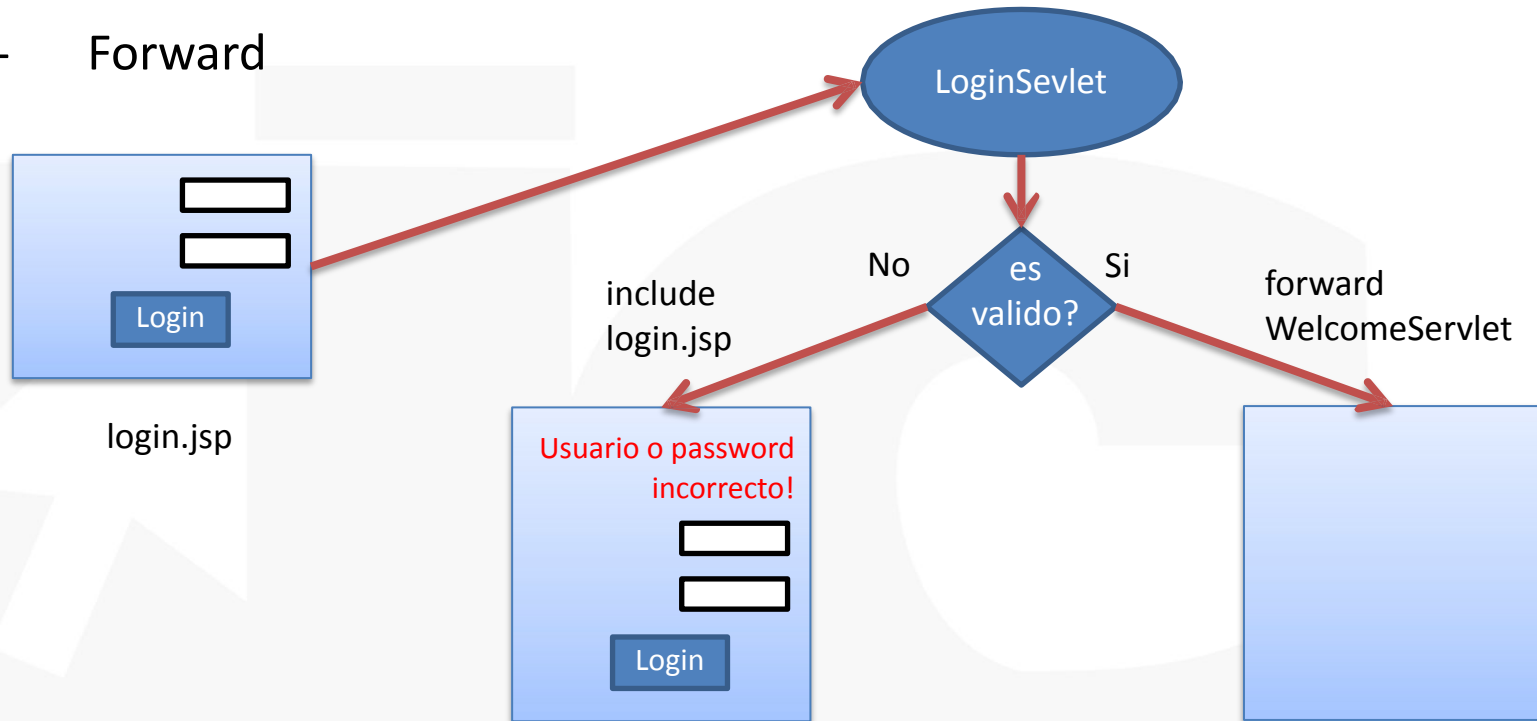
- Forward
- En el API Servlet, el objeto *RequestDispatcher* es el encargado de realizar el forward (o *include*) de un recurso en el servidor, ya sea un html, un jsp u otro servlet.

```
request.getRequestDispatcher("WelcomeServlet").forward(request, response);  
ó  
request.getRequestDispatcher("/WEB-INF/views/login.jsp").include(  
request, response);
```

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (c)

- Forward



vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (d)

- SendRedirect
- El método `sendRedirect` (*HttpServletResponse*) es el encargado de redirigir el *request* hacia otro recurso, forzando al cliente (navegador web) realizar una nueva petición HTTP hacia el recurso indicado.

```
response.sendRedirect("welcome");
```

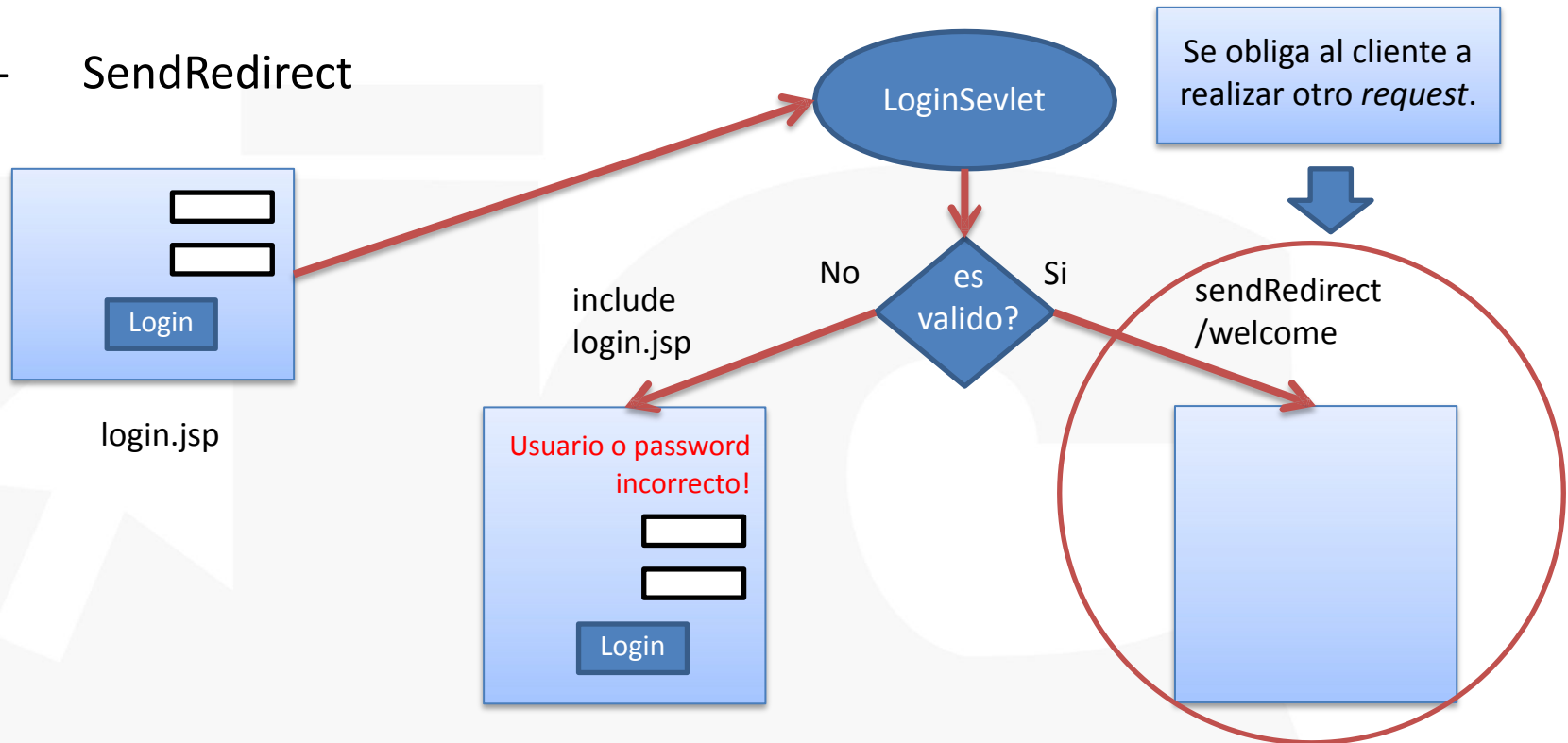
considerando que `WelcomeServlet` esta mapeado a la url `"/welcome"`

```
@WebServlet(name = "WelcomeServlet", urlPatterns = { "/welcome" })  
public class WelcomeServlet extends HttpServlet { ... }
```

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (e)

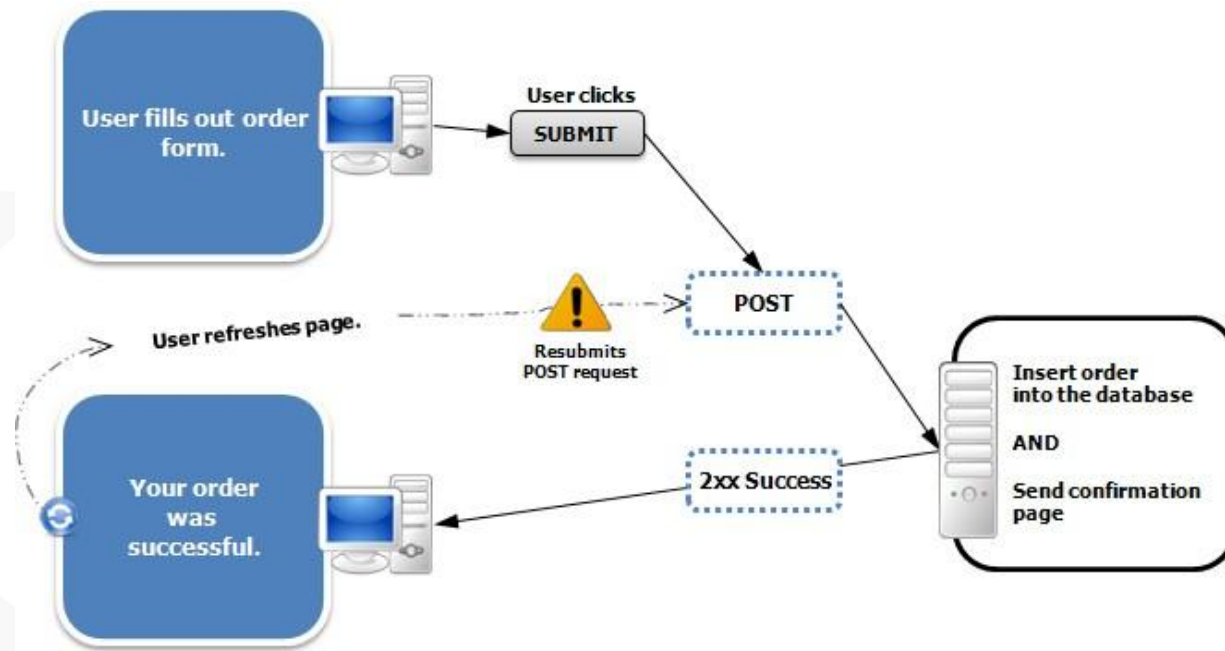
- SendRedirect



vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (f)

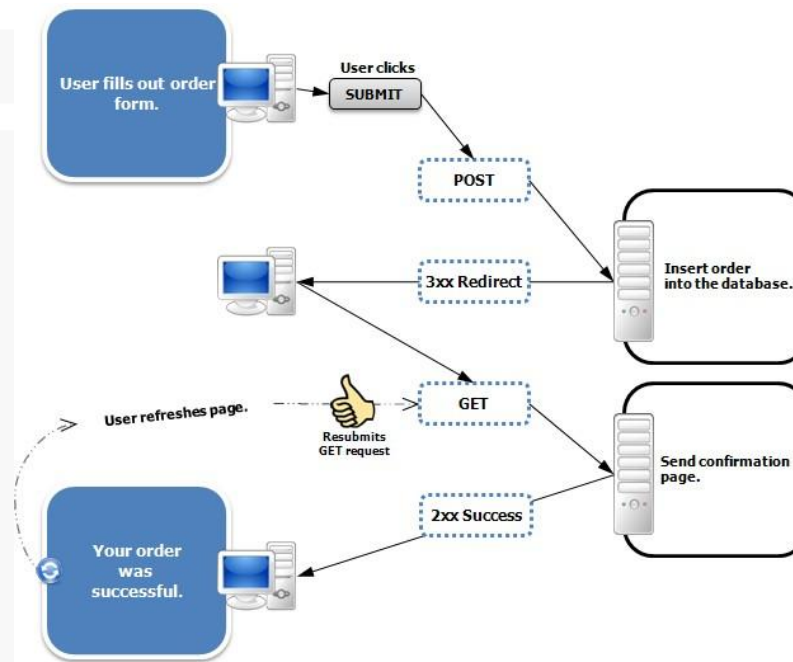
- Forward vs sendRedirect: Forward



vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

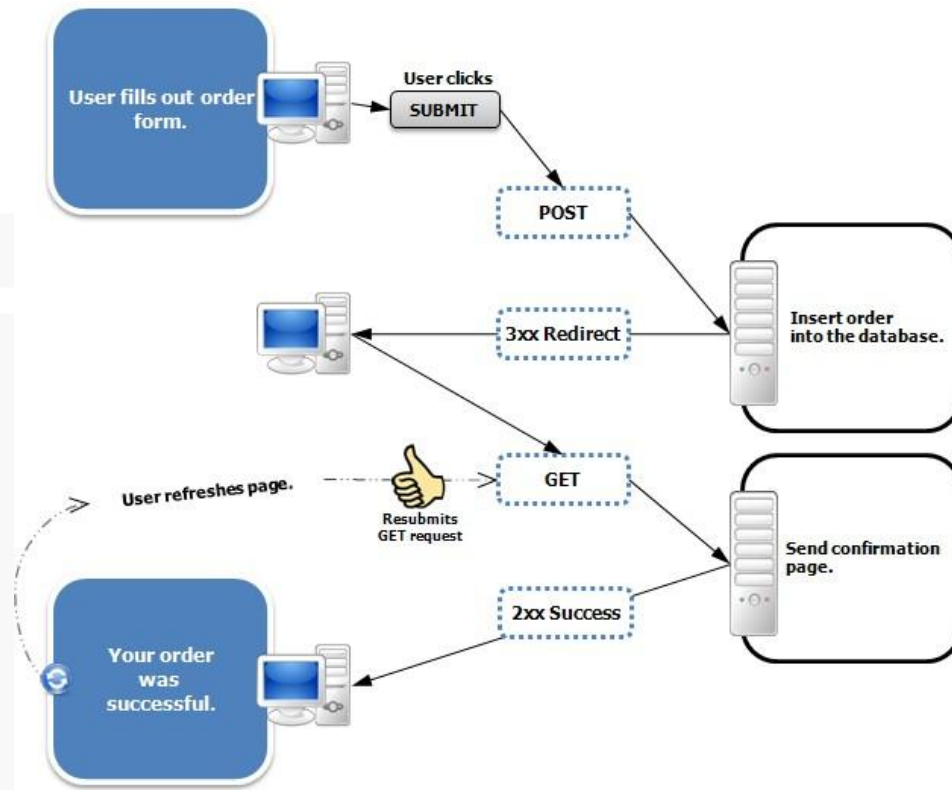
vi.v Formularios y redirección (g)

- Forward vs sendRedirect:
SendRedirect



vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (h)



vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



v. Formularios y redirección

- a. Tags Spring
- b. Model Interface
- c. @ModelAttribute
- d. Forward y sendRedirect
- e. @SessionAttributes

Práctica 29. Envío y recepción de formularios

vi.v Formularios y redirección (a)

- @SessionAttributes
- La anotación @SessionAttributes se utiliza, a nivel de controlador, para definir que modelos deberán ser almacenados en sesión.
- Es posible utilizar el objeto HttpSession del API Servlet directamente y almacenar y recuperar objetos de él sin embargo, no es recomendable.
- @SessionAttributes lista los nombres de los modelos que deben ser manejados en la sesión (o algún repositorio conversacional) del que se puedan recuperar datos.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (b)

- @SessionAttributes: Almacenar objetos en la sesión
- Agregar el objeto al modelo y listar el nombre del modelo (nombre de la variable del modelo) en la anotación @SessionAttributes.

@Controller

@SessionAttributes({ "userSession" })

public class **LoginController** {

 @RequestMapping(value = { "/login", "" }, method = RequestMethod.POST)

 public String login(**Model model**, @ModelAttribute User user) {

model.addAttribute("userSession", user);

 return "redirect:/admin/customers";

 }

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

vi.v Formularios y redirección (c)

- @SessionAttributes: Recuperar objetos de la sesión
- Utilizar @ModelAttribute sobre algún parámetro del *handler method*.

@Controller

@SessionAttributes({ "userSession" })

public class **AdminController** {

 @RequestMapping(value = "/admin/customers", method = RequestMethod.GET)

 public String showCustomers(Model model, **@ModelAttribute User userSession**) {

 return "admin_all_customers";

 }

}

<div id="topmenu">

 Welcome ***\${userSession.username}***

</div>

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



v. Formularios y redirección

- a. Tags Spring
- b. Model Interface
- c. @ModelAttribute
- d. Forward y sendRedirect
- e. @SessionAttributes

Práctica 29. Envío y recepción de formularios



vi.v Formularios y redirección. Práctica 29. (a)

- Práctica 29. Envío y recepción de formularios
- Implementar el envío de un formulario (comando) a la vista que presentará el formulario para después procesarlo.
- Implementar forward y sendRedirect.
- Analizar las diferencias entre forward y sendRedirect.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



Resumen de la lección

vi.v Formularios y redirección (a)

- Analizamos que existen diferentes Tags de Spring y su forma de incluirlas en las vistas JSP.
- Comprendimos como funciona el *binding* de objetos a través de comandos desde la vista hacia el controlador y viceversa.
- Comprendimos como agregar valores al modelo y como recuperarlos en las vistas.
- Comprendimos como crear un formulario básico en Spring MVC y cómo obtener sus valores.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección

Resumen de la lección

vi.v Formularios y redirección (b)

- Analizamos en que casos puede ser conveniente devolver un ModelAndView en lugar de un String desde los controladores.
- Comprendimos el uso y aplicación de la anotación @ModelAttribute.
- Comprendimos la diferencia entre los mecanismos forward y.sendRedirect.
- Comprendimos como se implementan las sesiones en Spring MVC.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



Esta página fue intencionalmente dejada en blanco.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



vi.vi **Validaciones**

Objetivos de la lección

vi.vi Validaciones

- Revisar como implementa Spring su API de Bean Validation.
- Comprender para qué se utiliza y como se implementa la Interface Validator.
- Implementar validación de objetos mediante el API de Spring Validation.
- Comprender la utilidad del objeto WebDataBinder y como se configura mediante la anotación @InitBinder en Spring MVC.
- Implementar validación de formularios mediante el API Spring Validation.

vi. Fundamentos Spring MVC y Spring REST - vi.v Formularios y redirección



vi. Validaciones

- a. Validator Interface
- b. @InitBinder

Práctica 30 – Parte 1. Spring Validation

Práctica 30 – Parte 2. Validación de formularios con Spring Validation

vi.vi Validaciones (a)

- Spring Framework implementa soporte para validación de beans (POJOs) mediante el API Spring Validation, mismo que proporciona soporte completo para la especificación JSR 303 Bean Validation.
- Es posible validar anotaciones JSR 303 sobre atributos de clases POJO tales como @NotNull, @Size, @Min, @Max, entre otros (no cubiertos en este curso).
- La implementación propia del API Spring Validation será la revisada en este curso.

vi. Fundamentos Spring MVC y Spring REST - vi.vi Validaciones



vi. Validaciones

a. Validator Interface

b. @InitBinder

Práctica 30 – Parte 1. Spring Validation

Práctica 30 – Parte 2. Validación de formularios con
Spring Validation

vi.vi Validaciones (a)

- Validator Interface.
- Spring Framework define la interface *org.springframework.validation.Validator* la cual esta totalmente desacoplada de cualquier infraestructura o contexto.
- Es posible utilizarla en cualquier contexto o capa, ya sea de negocio, presentación, DAO, etcétera.

vi.vi Validaciones (b)

- Validator Interface.
- La interface Validator utiliza un objeto Errors en donde se reportarán los errores de validación incurridos en el objeto target validado.

```
public interface Validator {  
    boolean supports(Class<?> clazz);  
    void validate(Object target, Errors errors);  
}
```

vi.vi Validaciones (c)

- Validator Interface: Ejemplo.

@Component

```
public class PersonValidator implements Validator {  
    @Override  
    public boolean supports(Class<?> clazz) {  
        return clazz.isAssignableFrom(Person.class);  
    }  
}
```

@Override **validate**(Object target, Errors errors) {

ValidationUtils.rejectIfEmptyOrWhitespace(errors, "age", "age.required", "Age is required.");

Person person = (Person) target;

if (person.getAge() != null && person.getAge() < 18) {

errors.rejectValue("age", "age.gt18", new Object[] { 18 }, "Age must be greather or equal than 18.");

}

}

}

```
public class Person {  
    private String name;  
    private Integer age;  
    // getters y setters  
}
```

vi. Fundamentos Spring MVC y Spring REST - vi.vi Validaciones

vi.vi Validaciones (d)

- Validator Interface.
- Para aplicar la validación sobre un objeto POJO dado es necesario hacer uso de las clases BindException y ValidationUtils.

```
Person person = new Person();
```

```
BindException errors = new BindException(person, Person.class.getName());
```

```
ValidationUtils.invokeValidator(personValidator, person, errors);
```

```
if (errors.hasErrors()) {  
    // obtener errores  
}
```



vi. Validaciones

a. Validator Interface

b. @InitBinder

Práctica 30 – Parte 1. Spring Validation

Práctica 30 – Parte 2. Validación de formularios con
Spring Validation

vi.vi Validaciones (a)

- @InitBinder
- La anotación @InitBinder se aplica sobre métodos no *handler methods*, sirve para configurar el *WebDataBinder* subtipo de *DataBinder*.
- Hasta el momento hemos comprendido que Spring maneja el *binding* entre los modelos de la vista y los controladores, sin embargo ¿Cómo lo hace? Ese es el trabajo del *WebDataBinder*.

vi.vi Validaciones (b)

- WebDataBinder
- A grandes rasgos el WebDataBinder es el objeto que se encarga de realizar el *binding* entre formularios a clases (objetos) POJO en el modelo, mismos que pueden ser recuperados a través de @ModelAttribute.
- WebDataBinder por defecto puede recuperar y mapear datos de tipo String, numérico (*Wrappers* o primitivos), y fecha (en el formato por defecto 'yyyy-mm-dd').

vi.vi Validaciones (c)

- WebDataBinder
- ¿Cómo podemos recuperar una fecha, de un formulario, y obtenerla, de el modelo, en un *handler method* con un formato diferente?
- ¿Cómo podemos recuperar tipos de dato más complejos que simples Strings, números y fechas?
- Es necesario configurar el WebDataBinder para que éste pueda *bindear* de forma correcta los modelos provenientes del cliente.

vi.vi Validaciones (d)

- WebDataBinder
- WebDataBinder se encarga de:
 - Registrar formateadores de tipos de datos personalizados.
 - Registrar *property editors*.
 - Registrar validadores.

vi.vi Validaciones (e)

- @InitBinder: Registro de Validadores

```
@Controller
@RequestMapping(value = "/persons")
public class PersonController {

    @Autowired
    private PersonValidator personValidator;

    @InitBinder(value = "createPersonForm")
    protected void configureInitBinderCreatePersonForm(WebDataBinder binder) {
        binder.setValidator(personValidator);
    }

    ...
}
```

vi.vi Validaciones (f)

- @InitBinder: Validando el modelo mediante @Valid o @Validated

```
@RequestMapping(value = "/create", method = RequestMethod.GET)
public String showCreatePersonPage(Model model) {
    model.addAttribute("createPersonForm", new Person());
    return "person_create_form";
}
```

```
<form:form id="form" action="persons/createPerson"
            commandName="createPersonForm" method="POST">
    Name: <form:input path="name" size="30" /> <form:errors path="name" cssClass="error" /> <br />
    Age: <form:input path="age" size="30" /> <form:errors path="age" cssClass="error" /> <br />

    <input class="button" type="submit" value="Create" />
</form:form>
```

person_create_form.jsp

vi.vi Validaciones (g)

- @InitBinder: Validando el modelo mediante @Valid o @Validated

```
@RequestMapping(value = "/createPerson", method = RequestMethod.POST)
public String createPerson(Model model,
    @Validated @ModelAttribute("createPersonForm") Person person, BindingResult result) {

    if (result.hasErrors()) {
        return "person_create_form";
    }
    ...
    return "redirect:persons";
}
```



vi. Validaciones

- a. Validator Interface
- b. @InitBinder

Práctica 30 – Parte 1. Spring Validation

Práctica 30 – Parte 2. Validación de formularios con
Spring Validation



vi.vi Validaciones. Práctica 30. (a)

- Práctica 30 – Parte 1. Spring Validation
- Implementar validación de beans programática mediante `ValidationUtils.invokeValidator` del API Spring Validation (proyecto 5-spring-web-mvc-hello-world).
- Comprender el mecanismo de validación de beans.

vi. Fundamentos Spring MVC y Spring REST - vi.vi Validaciones



vi. Validaciones

- a. Validator Interface
- b. @InitBinder

Práctica 30 – Parte 1. Spring Validation

Práctica 30 – Parte 2. Validación de formularios con Spring Validation

vi.vi Validaciones. Práctica 30. (a)

- Práctica 30 – Parte 2. Validación de formularios con Spring Validation
- Implementar validación de formularios de forma automática mediante @InitBinder (proyecto 5-spring-web-mvc-hello-world).
- Comprender el mecanismo de validación de beans.
- Implementar validación de formularios mediante @InitBinder y @Valid o @Validated.



Resumen de la lección

vi.vi Validaciones

- Comprendimos como se implementa el API Spring Validation en aplicaciones standalone y web con Spring MVC.
- Implementamos validación de formularios.
- Analizamos la utilidad de la anotación @InitBinder.
- Comprendimos la utilidad del objeto WebDataBinder.



Esta página fue intencionalmente dejada en blanco.



vi.vii Spring REST



Objetivos de la lección

vi.vii Spring REST (a)

- Comprender que es REST y cuales son algunas de sus ventajas frente a SOAP.
- Comprender los principios de REST y como implementar servicios REST bien formados.
- Analizar el termino HATEOAS.
- Analizar algunos ejemplos REST.
- Comprender como enviar diferentes tipos de código de status de peticiones HTTP en Spring MVC.

Objetivos de la lección

vi.vii Spring REST (b)

- Comprender como se configura, a nivel de dependencias, la implementación de servicios web REST en Spring MVC.
- Implementar servicios web REST mediante `@ResponseBody`, `@RequestBody` y `@RestController`.
- Analizar como mejorar la implementación de controladores mediante aspectos utilizando las anotaciones `@ControllerAdvice`, `@ExceptionHandler` y `@ResponseStatus`.



vii. **Spring REST**

- a. ¿Qué es REST?
- b. Principios de REST
- c. @RequestBody y @ResponseBody
- d. @RestController
- e. Código de status HTTP

Práctica 31. Implementación Servicios REST



vi.vii Spring REST (a)

- ¿Qué es REST?
- REST (*Representation State Transfer*), es un tipo de arquitectura o estilo arquitectónico para el desarrollo de servicios web que se apoya totalmente en el estándar HTTP.
- REST es un estilo arquitectónico totalmente desacoplado de alguna tecnología en particular.
- REST permite desarrollar arquitecturas distribuidas, al igual que SOAP Web Services.

vi.vii Spring REST (b)

- ¿Qué es REST?
- REST trabaja con entidades de negocio, e implementa las operaciones básicas de acceso a datos tales como el CRUD (*Create, Retrieve, Update y Delete*) mediante la aplicación del protocolo HTTP.

RESTful API
GET PUT POST DELETE

vi.vii Spring REST (c)

- ¿Qué es REST?
- Por lo general los desarrolladores no se preocupan por la interoperabilidad de la aplicación en desarrollo hacia con otras aplicaciones.
- REST es una alternativa más ligera, flexible y económica para la implementación de servicios web.



vi.vii Spring REST (d)

- ¿Qué es REST?
- REST se definió en el año 2000 por Roy Thomas Fielding, co-autor de la especificación HTTP.
- *“Podríamos considerar REST como un framework para construir aplicaciones web respetando HTTP”.*

vi.vii Spring REST (e)



- ¿Qué es REST?
- REST permite a cualquier aplicación que pueda ejecutar peticiones HTTP e interpretar texto (JSON), pueda consumir un servicio web sin necesidad de implementar protocolo SOAP.
- REST es el estilo arquitectónico más natural y estándar para crear servicios web REST (APIs REST).
- REST utiliza JSON (*JavaScript Object Notation*) como formato de intercambio de datos, aunque también puede implementar XML.

vi.vii Spring REST (f)

- ¿Qué es REST?
- REST utiliza JSON (*JavaScript Object Notation*) como formato de intercambio de datos, aunque también puede implementar XML.



```
{ "menu": { "id": "file", "value": "File",  
  "popup": {  
    "menuitem": [  
      { "value": "New", "onclick": "CreateNewDoc()" },  
      { "value": "Open", "onclick": "OpenDoc()" },  
      { "value": "Close", "onclick": "CloseDoc()" }  
    ]  
  }  
}}
```



```
<menu id="file" value="File">  
  <popup>  
    <menuitem value="New" onclick="CreateNewDoc()" />  
    <menuitem value="Open" onclick="OpenDoc()" />  
    <menuitem value="Close" onclick="CloseDoc()" />  
  </popup>  
</menu>
```



vi.vii Spring REST (g)

http://localhost:8080/Json/SyncReply/Contacts

```
{
  - Contacts: [
    - {
      FirstName: "Demis",
      LastName: "Bellot",
      Email: "demis.bellot@gmail.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Jobs",
      Email: "steve@apple.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Ballmer",
      Email: "steve@microsoft.com"
    },
    - {
      FirstName: "Eric",
      LastName: "Schmidt",
      Email: "eric@google.com"
    },
    - {
      FirstName: "Larry",
      LastName: "Ellison",
      Email: "larry@oracle.com"
    }
  ]
}
```

http://localhost:8080/Xml/SyncReply/Contacts

```
<ContactsResponse xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Contacts>
    <Contact>
      <Email>demis.bellot@gmail.com</Email>
      <FirstName>Demis</FirstName>
      <LastName>Bellot</LastName>
    </Contact>
    <Contact>
      <Email>steve@apple.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Jobs</LastName>
    </Contact>
    <Contact>
      <Email>steve@microsoft.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Ballmer</LastName>
    </Contact>
    <Contact>
      <Email>eric@google.com</Email>
      <FirstName>Eric</FirstName>
      <LastName>Schmidt</LastName>
    </Contact>
    <Contact>
      <Email>larry@oracle.com</Email>
      <FirstName>Larry</FirstName>
      <LastName>Ellison</LastName>
    </Contact>
  </Contacts>
</ContactsResponse>
```


vi.vii Spring REST (h)

- ¿Qué es REST?
- A diferencia de SOAP, REST no es un protocolo y no requiere mayor tratamiento para implementar comunicación productor y consumidor.

SOAP

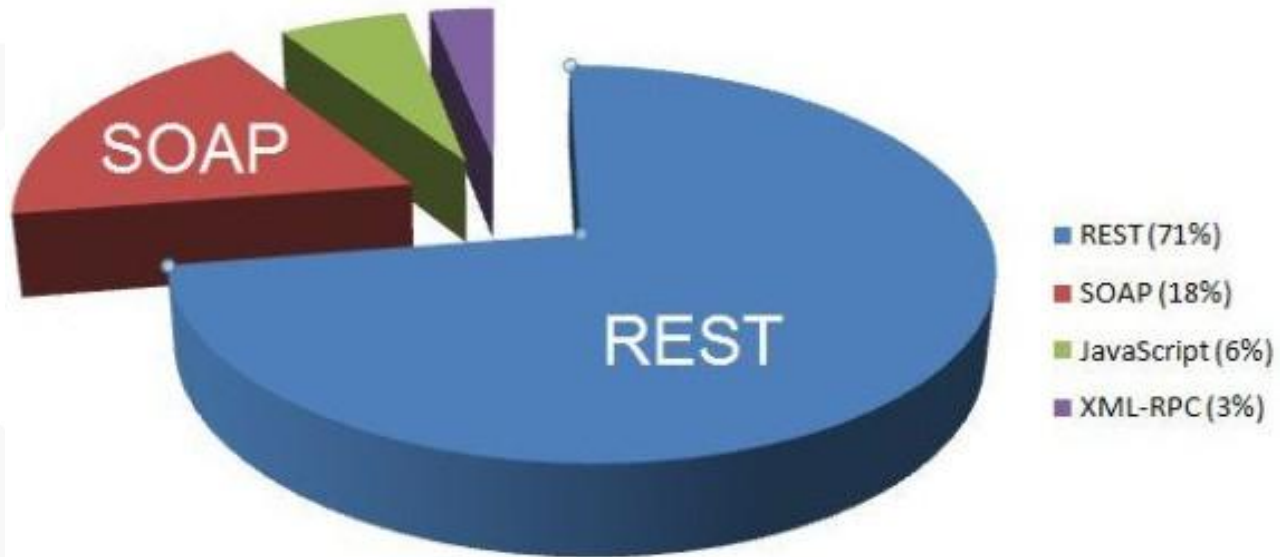


REST



vi.vii Spring REST (i)

- ¿Qué es REST?





vii. **Spring REST**

a. ¿Qué es REST?

b. Principios de REST

c. @RequestBody y @ResponseBody

d. @RestController

e. Código de status HTTP

Práctica 31. Implementación Servicios REST

vi.vii Spring REST (a)

- Principios de REST
- REST define como deben aplicarse estándares web como HTTP y URI para implementar servicios web.
- REST esta orientado a recursos, una entidad de negocio es un recurso, ejemplo:
 - Customer
 - Transfer
 - Book
 - Travel

vi.vii Spring REST (b)

- Principios de REST

1. Los recursos o entidades deben tener un identificador *Id* único.
2. Las entidades deben vincularse entre si mismas, aplicar hipermedia o también conocido como *HATEOAS* (*Hypermedia as the Engine of Application State*, Hipermedia como motor del estado de la aplicación).
3. Los recursos pueden tener diferentes representaciones multimedia (xml, json, pdf, XHTML, etc).

vi.vii Spring REST (c)

- Principios de REST

4. Aplicar los comandos HTTP correctamente:

- GET: Recuperar (*retrieve*)
- POST: Crear (*create*)
- PUT: Actualizar (*update*)
- DELETE: Eliminar (*delete*)
- PATCH: Actualización parcial (*patch*)

5. La comunicación entre el productor y consumidor no guarda estado (no sesiones, los servicios son Stateless).



vi.vii Spring REST (d)

- Principios de REST

6. Implementar correctamente la idempotencia.

- GET, PUT, DELETE, PATCH: Idempotente
- POST: No Idempotente

7. Implementar operaciones CRUD mediante RESTful y acciones mediante RESTless.

vi.vii Spring REST (e)

- Principios de REST
- REST no es:
 - Un protocolo
 - Un estándar
 - Un sustituto de SOAP



Follow the pattern

e.g. `http://{domain}/{type}/{concept}/{reference}`

Re-use existing identifiers

e.g. `http://education.data.gov.uk/id/school/123456`

Link multiple representations

e.g. `http://data.example.org/doc/foo/bar.html`

e.g. `http://data.example.org/doc/foo/bar.rdf`

Implement 303 redirects for real-world objects

e.g. `http://www.example.com/id/alice_brown`

Use a dedicated service

i.e. independent of the data originator

Avoid stating ownership

e.g. `http://education.data.gov.uk/ministryofeducation/id/school/123456` ❌

Avoid version numbers

e.g. `http://education.data.gov.uk/doc/school/v1/123456` ❌

Avoid using auto-increment

e.g. `http://education.data.gov.uk/id/school1/123456` ➡

e.g. `http://education.data.gov.uk/id/school1/123457` ➡

Avoid query strings

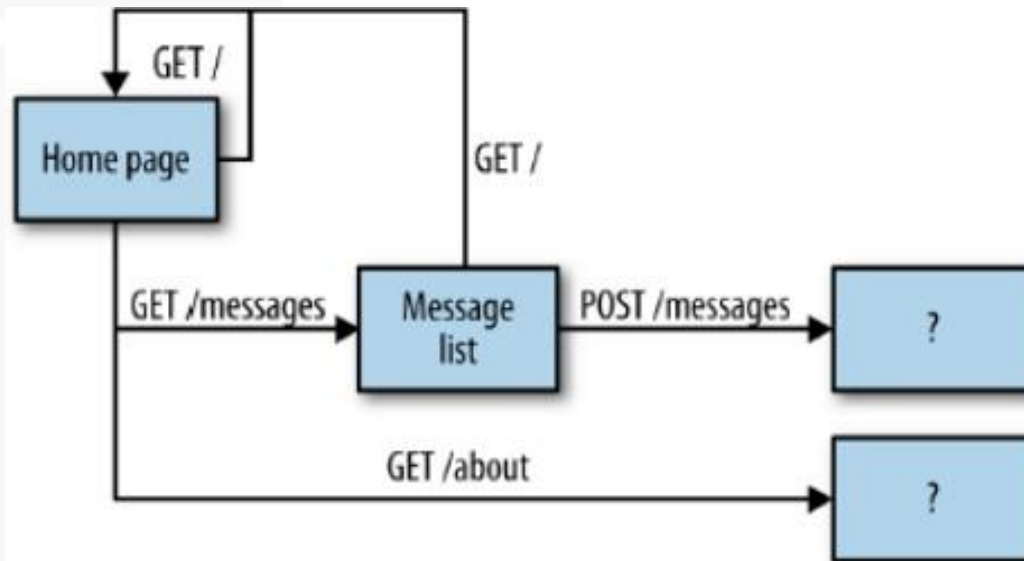
e.g. `http://education.data.gov.uk/doc/school?id=123456` ❌

Avoid file extensions

`http://education.data.gov.uk/doc/schools/123456.cs` ❌

vi.vii Spring REST (g)

- Ejemplo REST



vi.vii Spring REST (h)

- Ejemplo REST

<http://localhost:8080/api/books/>

GET – obtener todos los libros

POST – agregar un libro

<http://localhost:8080/api/books/{id}>

GET – obtener un libro

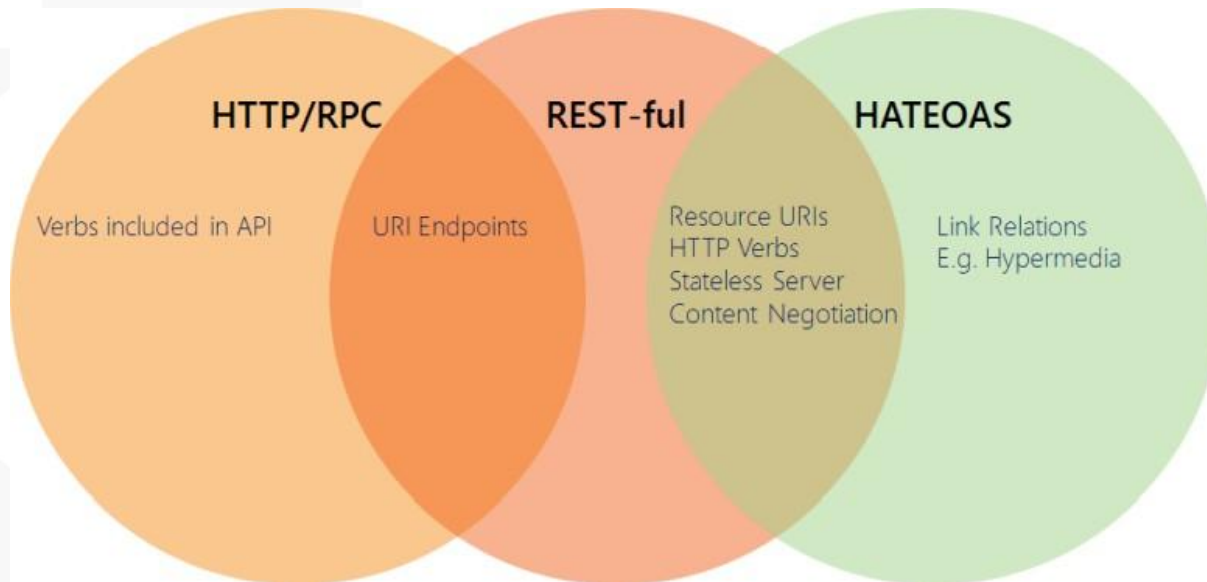
PUT – actualizar un libro (entidad completa)

PATCH – actualizar un libro (parcialmente)

DELETE – eliminar un libro

vi.vii Spring REST (i)

- HATEOAS



vi.vii Spring REST (j)

- HATEOAS como restricción REST

(a)

GET <https://api.fun.com>



*Movies: <https://api.fun.com/movies>
Music: <https://api.fun.com/music>
Account: <https://api.fun.com/account>*

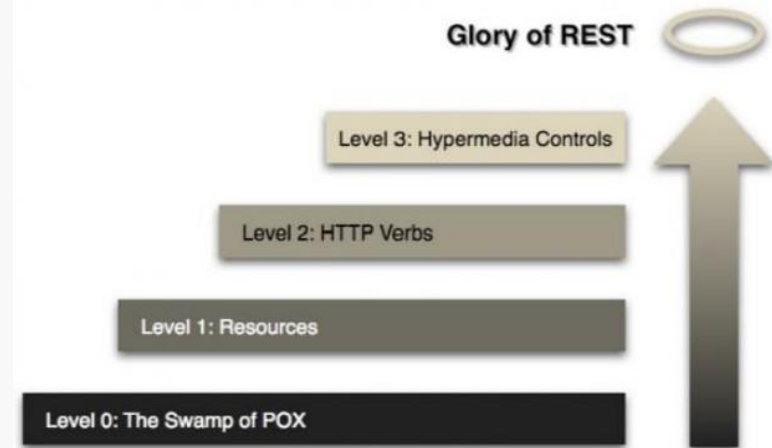
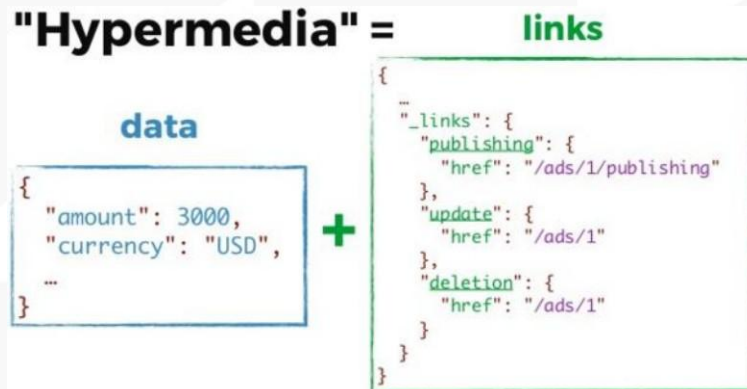
GET <https://api.fun.com/movies>



*Toy Story: <https://api.fun.com/movies/toy-story>
Wall-E: <https://api.fun.com/movies/wall-e>*

vi.vii Spring REST (k)

- HATEOAS como restricción REST (b)



```
[
  {
    "links": [
      {
        "rel": "self",
        "href": "http://localhost:8080/hateoas-demo/books/9780321356680"
      }
    ],
    "isbn": "9780321356680",
    "name": "Effective Java, 2nd Edition",
    "publisher": "Addison-Wesley",
    "publishingDate": "May 28, 2008",
    "author": {
      "links": [
        {
          "rel": "self",
          "href": "http://localhost:8080/hateoas-demo/authors/2"
        },
        {
          "rel": "books",
          "href": "http://localhost:8080/hateoas-demo/authors/2/books"
        }
      ],
      "authorId": 2,
      "name": "Joshua Bloch"
    }
  }
]
```



vii. Spring REST

- a. ¿Qué es REST?
- b. Principios de REST
- c. **@RequestBody y @ResponseBody**
- d. @RestController
- e. Código de status HTTP

Práctica 31. Implementación Servicios REST

vi.vii Spring REST (a)

- @RequestBody y @ResponseBody
- @RequestBody es una anotación que permite convertir el *body* o *payload* de la petición REST HTTP a un objeto mediante la implementación de `HttpMessageConverter`.
- @RequestBody se aplica sobre algún parámetro en un *handler method*.

vi.vii Spring REST (b)

- @RequestBody y @ResponseBody
- @ResponseBody es una anotación que permite convertir el objeto de retorno de un *handler method* a el *body* de la respuesta REST HTTP mediante la implementación de `HttpMessageConverter`.
- @ResponseBody se aplica sobre el método *handler method*. En muchas ocasiones se confunde su aplicación al tipo de retorno del *handler method*.

vi.vii Spring REST (c)

- @RequestBody y @ResponseBody
- En otras palabras, @RequestBody y @ResponseBody sirven para serializar y *deserializar* objetos.
- Permiten la serialización y deserialización de objetos delegando la conversión a un aspecto.
- Proveen soporte para implementar multimedia en servicios REST soportando distintos formatos (pdf, xml, json, etc).

vi.vii Spring REST (d)

- @RequestBody y @ResponseBody
- Spring MVC cuenta con diversas implementaciones `HttpMessageConverter`.
- La Interface `HttpMessageConverter` es responsable de convertir una petición HTTP a una clase específica (POJO) así como convertir una clase (POJO) al correspondiente *mime-type* solicitado.

vi.vii Spring REST (e)

- @RequestBody y @ResponseBody: Configuración
- Para poder convertir (serializar y *deserializar*) objetos de JSON a POJO y viceversa o, de XML a POJO y viceversa es necesario contar con:
 - **jackson-databind**: para serializar y *deserializar* objetos POJO a JSON y viceversa.
 - **jackson-dataformat-xml**: para realizar *marshall* y *unmarshall* de objetos POJO a XML y viceversa.

vi.vii Spring REST (f)

- @RequestBody y @ResponseBody: Configuración
- Contar con la librería jackson en el classpath habilita el uso automático de:
 - MappingJackson2HttpMessageConverter
 - MappingJackson2XmlHttpMessageConverter
- Spring MVC habilita los beans por defecto sin embargo es posible especificarlos mediante configuración XML o JavaConfig de requerirse alguna configuración especial, tal como habilitar *pretty print* para objetos JSON.

vi.vii Spring REST (g)

- @RequestBody y @ResponseBody: Ejemplo

```
@JacksonXmlElement(localName = "persons")  
public class Persons {
```

```
    @JacksonXmlElementWrapper(useWrapping = false)  
    @JacksonXmlProperty(localName = "person")  
    private List<Person> persons;
```

```
    public Persons(List<Person> persons) {  
        this.persons = persons;  
    }  
}
```

```
@JacksonXmlElement(localName = "person")  
public class Person {  
    private Integer id;  
    private String name;  
    private Integer age;  
}
```

vi.vii Spring REST (h)

- @RequestBody y @ResponseBody: Ejemplo

```
@Controller
@RequestMapping(value = "/persons")
public class PersonsController {

    private List<Person> persons = getPersons();

    @RequestMapping(value = { "/", "" }, method = RequestMethod.GET, produces = {
        MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
    public @ResponseBody Persons getAllPersons() {
        return new Persons(persons);
    }

    ...
}
```


vi.vii Spring REST (i)

- @RequestBody y @ResponseBody:

Ejemplo

```
...  
  
@RequestMapping(value =("/{id}", method = RequestMethod.GET, produces = {  
    MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })  
public @ResponseBody Person getPerson(@PathVariable Integer id) {  
    return persons.get(id - 1);  
}  
  
@RequestMapping(value = {"/", ""}, method = RequestMethod.POST, produces = {  
    MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })  
public void createPerson(@RequestBody Person person) {  
    persons.add(person);  
}  
}
```

Controller /persons

vi. Fundamentos Spring MVC y Spring REST - vi.vii Spring REST



vii. Spring REST

- a. ¿Qué es REST?
- b. Principios de REST
- c. @RequestBody y @ResponseBody
- d. @RestController
- e. Código de status HTTP

Práctica 31. Implementación Servicios REST

vi.vii Spring REST (a)

- @RestController
- A partir de la versión Spring 4 es posible utilizar la anotación @RestController como estereotipo de @Controller para definir la naturaleza de un controlador de tipo REST, la cual no requiere utilizar @ResponseBody en la firma de sus *handler methods*.
- Comúnmente se suele utilizar @Controller con *handler methods* que atienden peticiones HTTP y responden JSON o XML, o algún otro tipo de formato multimedia.

vi.vii Spring REST (b)

- @RestController
- Por conveniencia y para dar una mejor notación a las clases controladoras de servicios REST se sugiere utilizar la anotación @RestController en lugar de @Controller.
- @RestController ayuda a la intercepción de *handler methods* mediante aspectos.
- @RestController no omite la utilización de @RequestBody.

vi.vii Spring REST (c)

- @RestController: Ejemplo

```
@RestController
@RequestMapping(value = "/rest/persons")
public class PersonsRestController {

    private List<Person> persons = getPersons();

    @RequestMapping(value = { "/", "" }, method = RequestMethod.GET, produces = {
        MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })
    public Persons getAllPersons() {
        return new Persons(persons);
    }

    ...
}
```

vi.vii Spring REST (d)

- @RestController:

Ejemplo

```
...  
  
@RequestMapping(value =("/{id}", method = RequestMethod.GET, produces = {  
    MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE})  
public Person getPerson(@PathVariable Integer id) {  
    return persons.get(id - 1);  
}  
  
@RequestMapping(value = {"/", ""}, method = RequestMethod.POST, produces = {  
    MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE })  
public void createPerson(@RequestBody Person person) {  
    persons.add(person);  
}  
}
```

vi. Fundamentos Spring MVC y Spring REST - vi.vii Spring REST



vii. **Spring REST**

- a. ¿Qué es REST?
- b. Principios de REST
- c. @RequestBody y @ResponseBody
- d. @RestController
- e. Código de status HTTP

Práctica 31. Implementación Servicios REST

vi.vii Spring REST (a)

- Código de status HTTP
- Spring MVC permite devolver códigos de status de la solicitud HTTP, por ejemplo cuando un recurso no existe o cuando la respuesta no tiene contenido.
- ¿Qué debiera ocurrir si el recurso solicitado no existe?
 - ~~Lanzar una excepción y que el navegador muestre la traza de error.~~
 - Manejar la excepción y que el cliente obtenga una respuesta REST (JSON o XML) con el mensaje de la respuesta.

vi.vii Spring REST (b)

- Código de status HTTP
- No se recomienda que se lancen las excepciones hasta la capa de presentación, es necesario manejarlas en el servidor (entorno REST).
- Por lo regular, un API REST es auto navegable, es decir por medio de HATEOAS, el aplicativo cliente sabe que recursos puede solicitar.

vi.vii Spring REST (c)

- Código de status HTTP
- Existen múltiples estrategias para responder códigos de status HTTP diferente de 200 (Ok), es necesario validar y comprender las situaciones.

```
@RequestMapping(value = "/customers/{customerId}", method =  
RequestMethod.GET) public ResponseEntity<?> getCustomer(@PathVariable String  
customerId){  
    try{  
        Customer customer = customerService.findById(customerId); // única línea importante  
        return new ResponseEntity<>(customer, HttpStatus.OK);  
    }catch(Exception ex){  
    }  
    String errorMessage = "Customer not exists: "+ex.getMessage();  
    return new ResponseEntity<>(errorMessage, HttpStatus.BAD_REQUEST);  
}
```



vi.vii Spring REST (d)

- Código de status HTTP
- ¿Solución para evitar código intrusivo?

vi.vii Spring REST (e)

- Código de status HTTP
- ¿Solución para evitar código intrusivo?

Aspectos

vi.vii Spring REST (f)

- Spring MVC y Aspectos
- @ControllerAdvice y @RestControllerAdvice (Spring 4.3)
- La anotación @ControllerAdvice (pseudo-estereotipo @Component) permite su detección mediante el escaneo de paquetes (*component-scan*) configurando la clase anotada como un bean y habilita la definición de un aspecto para los *handler methods* de los controladores especificados.

vi.vii Spring REST (g)

- @ControllerAdvice
- Una clase (componente) anotada con @ControllerAdvice permite configurar:
 - Manejadores de excepciones.
 - Configuración de WebDataBinder, y
 - Establecer objetos (genéricos) al Modelo.

vi.vii Spring REST (h)

- @ControllerAdvice: Ejemplo Manejo de excepciones (a)

@ControllerAdvice

```
public class ExceptionControllerAdvice {
```

```
    @ExceptionHandler(Exception.class)
```

```
    public ModelAndView exception(Exception e) {  
        ModelAndView mav = new ModelAndView("exception");  
        mav.addObject("name", e.getClass().getSimpleName());  
        mav.addObject("message", e.getMessage());  
        return mav;  
    }  
}
```

vi.vii Spring REST (i)

- @ControllerAdvice: Ejemplo Manejo de excepciones (b)

```
@ControllerAdvice(assignableTypes = { PersonsController.class })  
public class ExceptionRestControllerAdvice {  
  
    @ExceptionHandler(RuntimeException.class)  
    @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)  
    @ResponseBody  
    public RestResponseError onException(RuntimeException re) {  
        RestResponseError restResponseError = new RestResponseError(  
            HttpStatus.INTERNAL_SERVER_ERROR, re.getMessage(),  
            re.getClass().getSimpleName());  
  
        return restResponseError;  
    }  
}
```


vi.vii Spring REST (j)

- Código de status HTTP
- La anotación `@ExceptionHandler` permite anotar un método como manejador de excepciones (comúnmente llamado *exception handler*).
- Un *exception handler* permite manipular cualquier excepción en ambiente web (mvc o rest) permitiendo responder algún modelo o vista específica.
- Es aplicable a cualquier método de clase controladora sin embargo se recomienda su uso en clases `@ControllerAdvice`.

vi.vii Spring REST (k)

- Código de status HTTP
- La anotación `@ResponseStatus` permite enviar un código de status de la petición HTTP al cliente.
- La anotación es aplicable a *handler methods* como a *exception handlers*.
- Por defecto se espera un correcto funcionamiento de los *handler methods*, por tanto no se espera devolver un código de status HTTP diferente de 200 (OK).



vi.vii Spring REST (I)

- Código de status HTTP
- Se recomienda la aplicación de `@ResponseStatus` en conjunto con `@ExceptionHandler` para poder habilitar el envío de un código de status HTTP (de error) cuando ocurra alguna excepción y ésta sea notificada al cliente.
- Nota: La anotación `@RestControllerAdvice` funciona exactamente igual que `@ControllerAdvice` pero agrega la funcionalidad de `@ResponseBody` a los métodos *exception handler* implementados.

vi.vii Spring REST (m)

- @RestControllerAdvice: Ejemplo (Spring 4.3, no cubierto en el curso)

```
@RestControllerAdvice(assignableTypes = { PersonsController.class })  
public class ExceptionRestControllerAdvice {  
  
    @ExceptionHandler(RuntimeException.class)  
    @ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)  
    public RestResponseError onException(RuntimeException re) {  
        RestResponseError restResponseError = new RestResponseError(  
            HttpStatus.INTERNAL_SERVER_ERROR, re.getMessage(),  
            re.getClass().getSimpleName());  
  
        return restResponseError;  
    }  
}
```



vii. **Spring REST**

- a. ¿Qué es REST?
- b. Principios de REST
- c. @RequestBody y @ResponseBody
- d. @RestController
- e. Código de status HTTP

Práctica 31. Implementación Servicios REST



vi.vii Spring REST. Práctica 31. (a)

- Práctica 31. Implementación Servicios REST
- Implementar servicios REST mediante @RequestBody y @ResponseBody.
- Implementar servicios REST mediante @RestController.
- Implementar mejores prácticas para la implementación de manejadores de excepciones en capa web mediante @ControllerAdvice.



Resumen de la lección

vi.vii Spring REST (a)

- Comprendimos los principios de REST así como el principio HATEOAS.
- Implementamos servicios REST con Spring MVC.
- Comprendimos la diferencia de utilizar `@Controller` y `@RestController`.
- Verificamos como enviar diferentes códigos de status de peticiones HTTP.
- Analizamos los beneficios de utilizar `@ControllerAdvice` para separar el encapsulamiento de manejo de excepciones.
- Realizamos pruebas a los servicios REST mediante un cliente REST independiente.



Esta página fue intencionalmente dejada en blanco.