Data Binding en Spring MVC

- **Data Binding** es el mecanismo mediante el cual Spring MVC **extrae dinámicamente** los datos de entrada del usuario y los asigna a objetos de **Modelo** de nuestra aplicación.
  - ✓ Un ejemplo de **Data Binding** en Spring MVC sería almacenar los datos de un formulario HTML a un objeto de Modelo (AUTOMÁTICAMENTE).
    - **AUTOMÁTICAMENTE** significa no hacerlo manualmente, por ejemplo:
      - **String name = request.getParameter("nombre")** → Como se hace comúnmente utilizando el API de Servlets.
      - **@RequestParam("nombre") String name** → Como lo hacemos en Spring MVC.

- Cuando utilizamos **Data Binding** en Spring MVC, la conversión de los tipos de datos se hace automáticamente.
  - ✓ Los parámetros de una petición HTTP son enviados como texto simple (String).
    - Spring MVC convierte los parámetros de la petición HTTP de tipo String, al tipo de dato según nuestra clase de modelo (int, Date, double, etc).

- La validación de los datos de entrada también se hace automáticamente.
  - ✓ Se puede crear una validación de datos personalizada.

Ejemplo de Data Binding

VIEW

```
<form th:action="@{/vacantes/save}" method="post">

  <input type="text" name="nombre" />


  <select name="estatus" >
      <option value="Creada">Creada</option>
      <option value="Aprobada">Aprobada</option>
  </select>


  <textarea name="detalles"></textarea>

 <button type="submit">Guardar</button>
</form>
```

MODEL

```
public class Vacante {

  private int id;

  private String nombre;

  private Date fecha;

  private String estatus;

  private String detalles;

  // getters / setters
}
```

CONTROLLER

```
@PostMapping("/save")
public String guardar(Vacante vacante){
    serviceVacantes.guardar(vacante);
    return "someView";
}
```

Al enviar el formulario (POST):

1. La petición la recibe el controlador.
2. Spring MVC extraé cada parámetro de la petición.
3. (Data Binding). Se asigna cada parámetro de la petición a cada propiedad del Modelo, pero solo si el nombre de un parámetro de la petición (input del formulario) coincide con el de una propiedad de nuestro objeto de modelo.

Anotación @InitBinder – Personalizar Data Binding

```
Error
Failed to convert from type [java.lang.String] to type [java.util.Date] for value '09-04-2019'; nested exception is
java.lang.IllegalArgumentException]
```

- El error anterior significa que Spring no puede convertir el valor del parámetro de la petición "09-04-2019" a un tipo Date.
  - ✓ La causa es que por Defecto Spring espera la fecha en el formato según este configurada en el sistema operativo donde este la aplicación.
- La anotación **@InitBinder** permite crear métodos para configurar el Data Binding directamente en el controlador.
- Los métodos marcados con la anotación @InitBinder no regresan ningún valor.
  - ✓ Normalmente son declarados como void.
- Comúnmente reciben un párametro de tipo WebDataBinder.
- El siguiente ejemplo muestra como utilizar @InitBinder para configurar CustomDateEditor para todas las propiedades de tipo java.util.Date

```
@InitBinder
public void initBinder(WebDataBinder webDataBinder) {
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
    webDataBinder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, false));
}
```

Control de errores en Data Binding

- Cuando Spring MVC realiza Data Binding pueden surgir errores. Por ejemplo:
  - ✓ Errores de formato. Por ejemplo el usuario ingresa la fecha 31-02-2017.
  - ✓ Errores de conversión. Por ejemplo en el atributo salario (private Double salario) el usuario ingresa un valor alfanumérico.
  - ✓ Omitir campos requeridos.
- Para revisar posibles errores después del Data Binding debemos agregar un parámetro de tipo BindingResult **INMEDIATAMENTE** después de la clase de Modelo.Ejemplo:

```
@PostMapping("/save")
public String guardar(Vacante vacante, BindingResult result) {
    serviceVacantes.guardar(vacante);
    return "someView";
}
```

- Después de agregar el parámetro BindingResult, podríamos verificar si hay errores y en caso de haber, lo correcto sería mostrárselos al usuario en la vista para su corrección.

BindingResult – Verificar si hay errores

- Verificar si hay errores:

```
@PostMapping(value = "/save")
public String guardar(Vacante vacante, BindingResult result) {
    if (result.hasErrors()) {
      return "vacantes/formVacante";
    }
    serviceVacantes.guardar(vacante);
    return "someView";
}
```

- Desplegar errores [consola]

```
for (ObjectError error: result.getAllErrors()){
   System.out.println("Ocurrio un error: " + error.getDefaultMessage());
}
```

BindingResult – Desplegar errores en la vista

- Para vincular los errores en la vista, también debemos enviar al formulario un objeto de nuestra clase de modelo (se declara como parametro):

```
@GetMapping("/create")
public String crear(Vacante vacante) {
   return "vacantes/formVacante";
}
```

- Vincular nuestro HTML FORM con el objeto de modelo (th:object)

```
<form th:action="@{/vacantes/save}" method="post" th:object="${vacante}">
```

- Utilizar la expresión **${#fields.hasErrors('*')}** en un condicional para preguntar si existieron errores al realizarse el Data Binding.
- En caso de existir errores, recorrer (th:each) la colección de errores existentes:

  ${#fields.errors('*')}

```
<div th:if="${#fields.hasErrors('*')}" class='alert alert-danger' role='alert'>
  Por favor corrija los siguientes errores:
  <ul>
     <li th:each="err : ${#fields.errors('*')}" th:text="${err}" />
  </ul>
</div>
```

Redirect y Flash Attributes

- **Los atributos flash** proporcionan una forma de almacenar atributos para poder ser usados en otra petición diferente.
- Comúnmente son usados cuando hacemos una **redirección** utilizando el patrón Post/Redirect/Get.
  - ✓ Ejemplo: **return "redirect:/vacantes/index";**
- Los atributos flash son almacenados temporalmente antes de hacer el redirect **(tipícamente en la sesión)** para tenerlos disponibles después del redirect. Despúes del Redirect son eliminados de la sesión automáticamente.

Ejemplo (Controlador)

```
@PostMapping("/save")
public String guardar(Vacante vacante, BindingResult result, RedirectAttributes attributes) {
    if (result.hasErrors()) {
      System.out.println("Existieron errores");
      return "vacantes/formVacante";
    }
   // Procesar objeto de modelo
   attributes.addFlashAttribute("msg", "Registro Guardado");
   return "redirect:/vacantes/index";
}
```

Ejemplo (Vista)

```
<div th:if="${msg != null}" class='alert alert-success' th:text="${msg}" role='alert'></div>
```

Registro Guardado

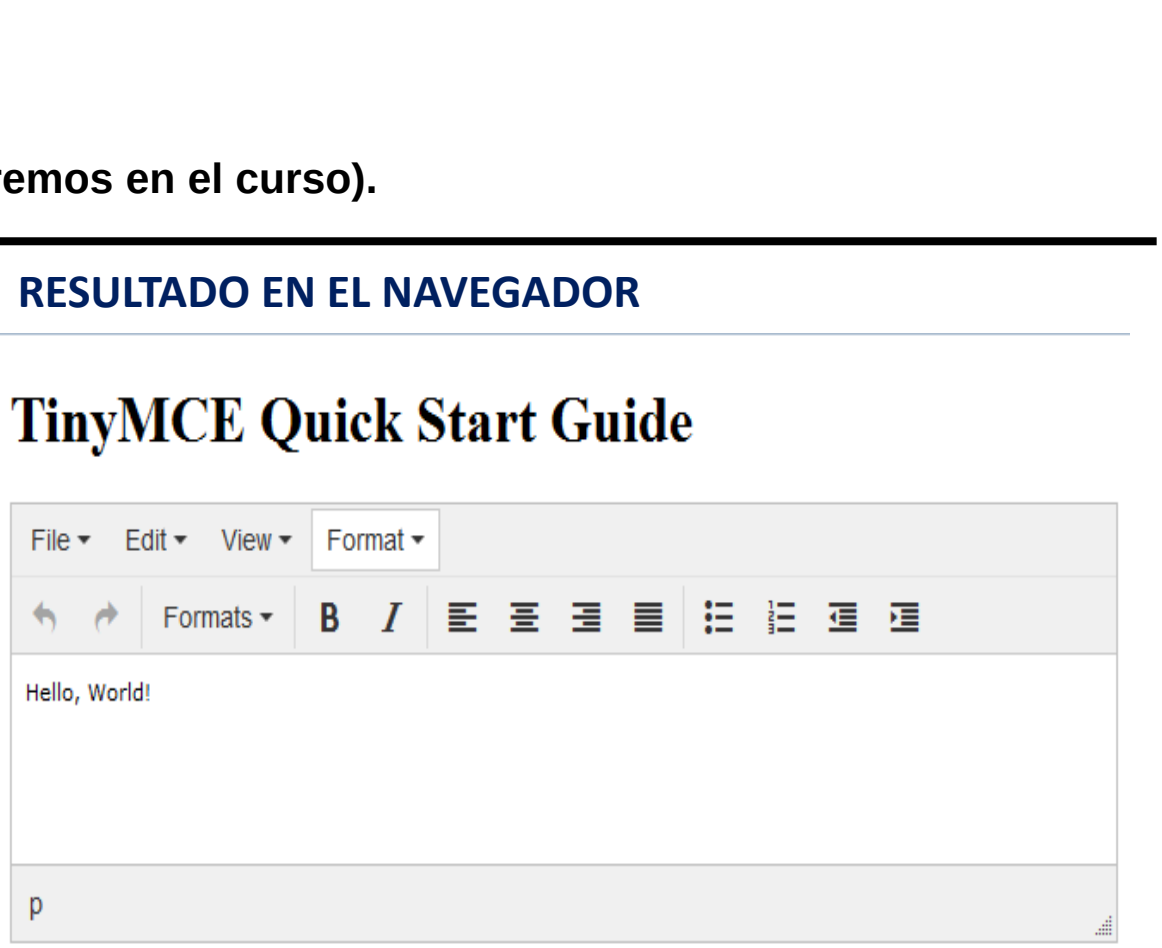
TinyMCE – Editor web (HTML-Javascript)

- Editor HTML basado en Javascript y HTML (independiente de la plataforma).
- Fácilmente convierte un elemento <textarea> en un editor HTML completamente funcional.
- Existe una versión Community y una versión de paga con más funcionalidad.
- Dos tipos de instalación (https://www.tinymce.com)
  - ✓ Vía CDN (Content Delivery Network). Se necesita Internet.
  - ✓ Descargar los archivos CSS, Javascript (esta forma la usaremos en el curso).

EJEMPLO CODIGO HTML

```
<!DOCTYPE html>
<html>
<head>
  <script src='https://cloud.tinymce.com/stable/tinymce.min.js'></script>
  <script>
    tinymce.init({  selector: '#mytextarea' });
  </script>
</head>
<body>
<h1>TinyMCE Quick Start Guide</h1>
  <form method="post">
    <textarea id="mytextarea">Hello, World!</textarea>
  </form>
</body>
</html>
```

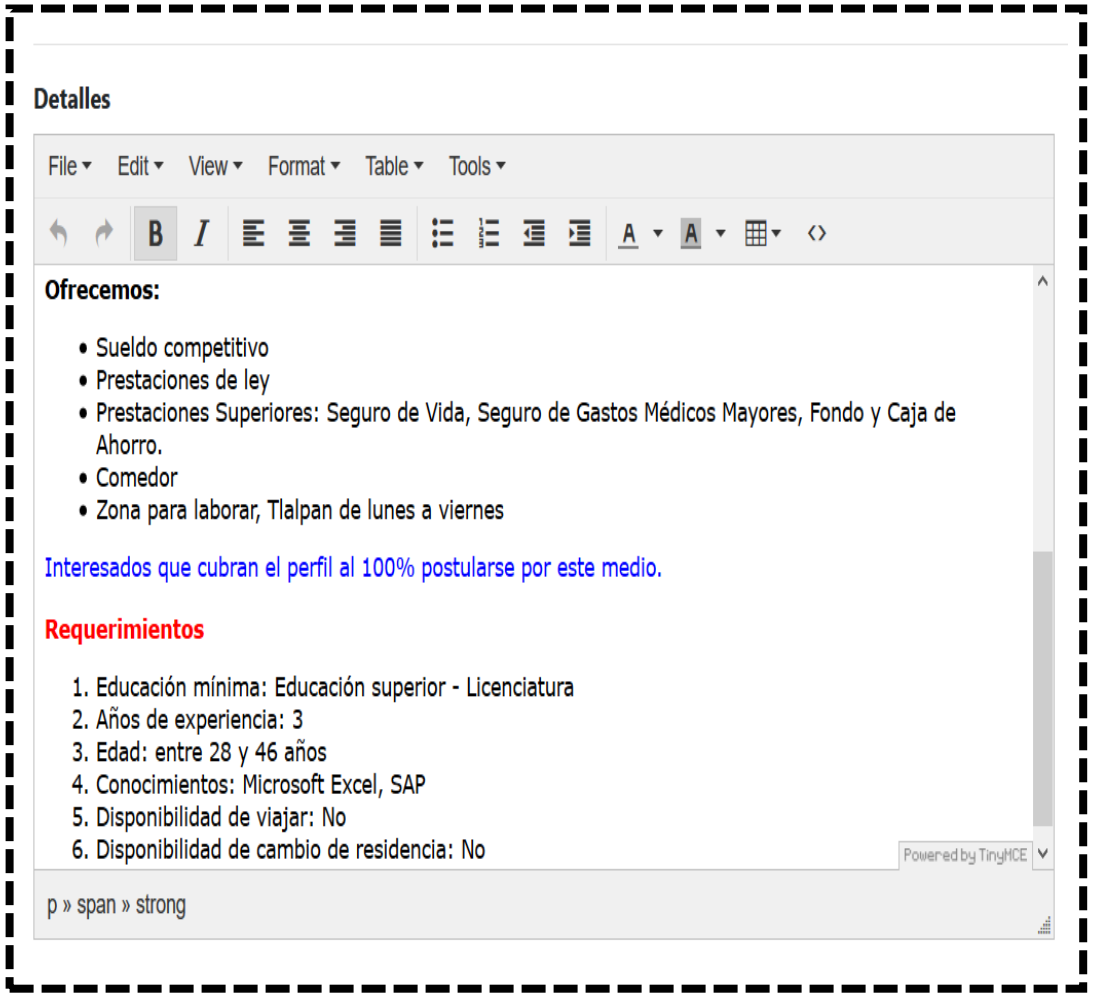
RESULTADO EN EL NAVEGADOR



TinyMCE – Formulario de creación de Noticias/Novedades

- El archivo formVacante.html de la plantilla HTML ya tiene incluido el plugin para el input <select> de los detalles (solo hay que activarlo).

Formulario HTML – Crear Vacante



La vacante renderizada en la vista de detalles (El formato se conserva)