|  |  |  |  |
| --- | --- | --- | --- |
| **CARRERA** | **SEMESTRE** | **CÓDIGO MATERIA** | **NOMBRE DE LA MATERIA** |
| Ingeniería Informática | 8º | 805 | Programación Web |

|  |  |  |
| --- | --- | --- |
| **PRÁCTICA N.:** | **TEMA:** | Validadores predefinidos ActionError, ActionMessage |
| 1  2  3  4 | **NOMBRES:** | Bryan Acosta |
| **FECHA ENTREGA:** | 17 de enero de 2019 |

|  |  |
| --- | --- |
| **1.** | **INTRODUCCIÓN** |
| **Framework Struts**  Antes de entrar en detalles sobre el funcionamiento de Struts vamos a ver primero la filosofía general de funcionamiento. Veremos también por qué usar Struts y las posibles alternativas.  Hay dos "generaciones" de Struts: la 1.x y la 2. El cambio de la generación 1 a la 2 no significa únicamente que se hayan añadido nuevas características. Es un cambio completo, tanto a nivel interno como externo. Cuando surgió Struts 1 era el único *framework* existente de este tipo y en su diseño original había deficiencias en cuanto a flexibilidad y simplicidad de uso. Todo esto se ha solucionado en la versión 2. No obstante, y de manera paradójica, Struts 2, aun siendo mucho más potente, flexible y fácil de usar, no ha tenido ni de lejos tanta difusión como la versión 1, simplemente porque ahora ya tiene la competencia de otros frameworks como Spring o JSF.  Veamos cómo implementa Struts los componentes del patrón Modelo-Vista-Controlador:   * El **controlador** es un servlet, de una clase proporcionada por Struts. Será necesario configurar la aplicación web (a través del fichero web.xml) para que todas las peticiones del usuario se redirijan a este servlet. * El controlador despacha las peticiones del usuario a la clase adecuada para ejecutar la **acción**. En struts, las clases que ejecuten las acciones deben *heredar de la clase* ***Action***. * La **vista** se implementará normalmente mediante páginas JSP. Struts ofrece dos herramientas para ayudar en la presentación de datos: los **ActionForms** son clases que capturan los datos introducidos en formularios y permiten su validación. Las **librerías de etiquetas** permiten mostrar errores y facilitar el trabajo con formularios.   La implementación del **modelo** corre enteramente a cargo del desarrollador, ya que es propio de la capa de negocio y no está dentro del ámbito de Struts | |
|  | |

|  |  |
| --- | --- |
| **2.** | **LOGRO DE APRENDIZAJE** |
| * Aprender el uso de validadores con el fin de suprimir código innecesario. * Creación de aplicaciones con un control de inserción de datos. | |

|  |  |
| --- | --- |
| **3.** | **MARCO TEÓRICO** |
| La utilización de validadores simplifica enormemente el proceso de validación de datos de usuario en una aplicación Struts al proporcionar un mecanismo de validación automática, de tipo declarativo, que evita la codificación de estas tareas desde código, permitiendo además la validación de los datos tanto en el cliente como en el servidor.  La utilización de validadores en una página JSP se apoya en el empleo de una serie de componentes que proporcionan toda la funcionalidad necesaria para llevar a cabo la validación automática y que habrá que configurar adecuadamente en cada aplicación, éstos componentes son:   * Plug-in validator: es la clase incorporada en el API de Struts encargada de gestionar las validaciones automáticas dentro de una aplicación Web * Archivos de configuración: Además del archivo de configuración de Struts, struts-config.xml, las aplicaciones que utilizan validadores necesitan dos archivos de configuración adicionales:   + 1. **validator-rules.xml:** El archivo validator-rules.xml, incluido en el paquete de distribución de Struts, contiene la declaración de estas clases de validación así como los métodos utilizados para realizar cada una de las rutinas de validación predefinidas, asociando a cada uno de estos métodos un nombre lógico que luego podrá ser utilizado para asignar una determinada regla de validación a un componente gráfico.     2. **validation.xml**: En este archivo de configuración el programador deberá asignar a cada campo cuyos datos quiera validar la regla o reglas de validación definidas en el archivo validator-rules.xml * La clase ValidatorForm: A la hora de definir la clase JavaBean en la que se recogerán los datos procedentes del formulario cliente y de cara a poder utilizar la validación automática proporcionada por los métodos definidos en validator-rules.xml, debemos utilizar como clase base ValidatorForm en vez de ActionForm. * Archivo de recursos ApplicationResource.properties: se utiliza en las aplicaciones Struts para almacenar cadenas de texto a las que se les asocia una clave, cadenas que pueden tener diferentes usos dentro de la aplicación. (Martin, 2007) | |
|  | |

|  |  |  |
| --- | --- | --- |
| **4.** | **PROCEDIMIENTO (DESCRIPCIÓN)** | |
| **A** | **HARDWARE REQUERIDO** | **SOFTWARE REQUERIDO** |
|  | * Procesador x86-x64 o AMD 2.5GHz + * RAM 4GB+(mínimo) * GPU 128MB+ | * Windows 7 o superior * MySQL * Web Browser: Google Chrome 30+, Firefox 24+, Opera 20+, Edge, IE12+ * NetBeans IDE 8.1 * Glassfish |
| **B** | **DESARROLLO DE LA PRÁCTICA** | |
|  | **Práctica 1: Web Control Datos Registro**  Creación de un nuevo proyecto Web Application, con la única diferencia de escoger como framework Struts 2.0. cabe recalcar que se debe agregar un pluggin adicional para poder escoger esta opción.  Se creará la siguiente estructura de archivos.    En la clase ValidacionForm, la cual hereda de ValidatorForm ,se encapsularan los atributos con sus respectivos métodos get-set.    En este caso el controlador dirige a la vista llamada *autenticación*    Dentro de nuestro archivo ApplicationResource.properties validaremos los siguientes atributos, en el cual gestionaremos el mensaje a enviar.  En nuestro archivo de configuración redireccionaremos la funcionalidad de nuestra aplicación.  **Práctica 2. Web Control Datos Usuario.**   1. Realizamos la siguiente estructura de archivos.      1. Encapsulamos los atributos de nuestra aplicación en la clase RegistroForm2, la cual hereda de ActionForm. 2. El controlador RegistroAction2 redirige a nuestra vista *registro.jsp* 3. Dentro de *ApplicationResource.properties* controlamos que debe ir en cada cuadro te texto , y también controlamos que la cadena de longitud mínima debe ser de 4 caracteres.      1. Nuestra vista *registro.jsp* contendrá etiquetas html que muestras los cuadros de texto para la inserción de datos. 2. Como archivo final de configuración tenemos *struts-config.xml*  en el cual controlaremos la funcionalidad de nuestra aplicación.   **Práctica 3. API Struts-ActionErrors y ActionMessage**   1. Para la creación de la aplicación crearemos la siguiente estructura de archivos.      1. Se empieza creando el modelo, la clase RegistroForm que a diferencia de las anteriores, y obviamente encapsula los atributos que involucran la lógica de negocio, es necesario contralar, errores personalizados por los campos que deseemos gestionar. En este caso usuario, password, y el email.      1. Como controlador tenemos la clase *RegistroAction.java*  que nos permite la redirección hacia la vista *registro.* 2. Dentro de nuestro archivo *ApplicationResource.properties* establecemos los mensajes personalizados a mostrar al usuario , como en este caso son :      1. Como vista se añaden ciertas etiquetas que permite mostrar los mensajes establecidos en el anterior archivo.      1. Finalmente tenemos struts-config, el cual controlaremos la funcionalidad de nuestra aplicación.     **Práctica 3. API Struts-ActionErrors y ActionMessage**   1. Para la creación de la aplicación crearemos la siguiente estructura de archivos.      1. Se empieza creando el modelo, la clase RegistroForm que a diferencia de las anteriores, y obviamente encapsula los atributos que involucran la lógica de negocio, es necesario contralar, errores personalizados por los campos que deseemos gestionar. En este caso usuario, password, y el email. 2. Como controlador tenemos la clase RegistroAction.java que nos permite la redirección hacia la vista registro. 3. Finalmente tenemos struts-config, el cual controlaremos la funcionalidad de nuestra aplicación. | |
|  | **RESULTADOS DE LA PRÁCTICA** | |
|  | **Práctica 1:**    **Práctica 2:**    **Práctica 3:**    **Práctica 4:** | |

|  |  |
| --- | --- |
| **5.** | **CONCLUSIONES Y RECOMENDACIONES** |
| ***Conclusiones:***   * El ahorro de código es una característica importante a la hora de crear una aplicación web, es lo que nos ofrece Struts, el ahorro de líneas de código para hacer validaciones en nuestras aplicaciones.   ***Recomendaciones:***   * Encapsular los atributos que van a ser almacenados en los Beans, para que no sean modificados y cumplir los estándares de programación. * Bajar el firewall para que no exista problemas al momento de ejecutar nuestras aplicaciones. | |

|  |  |
| --- | --- |
| **6.** | **ANEXOS** |
| **Anexo 1:** Añadir el plugin necesario para poder trabajar con el framework struts 2.0    Se adjuntará un CD con el proyecto generado para la correspondiente revisión por parte del Ingeniero. | |

|  |  |
| --- | --- |
| **7.** | **REFERENCIAS** |
| Bibliografía Martin, A. J. (2007). *Struts 2.0 .* Madrid: RA-MA. | |

|  |  |
| --- | --- |
| **DICTADO POR:** | Julio César Mendoza Tello |