Curso Spring – RichFaces – Lo esencial de RichFaces

Marzo 2011 – JoeDayz

Susan Inga

Contenido

- ✓ Introducción
- ✓ ¿Qué aprenderemos?
- ✓ Ganaremos experiencia en…
- ✓ Introducción a RichFaces
- ✓ Características principales
- ✓ Configurando la infraestructura de RichFaces
- ✓ Paso 1: Evaluando el estado inicial de la aplicación web
- ✓ Paso 2: Verificando la configuración del backend
- ✓ Paso 3: Configurando el RichFaces Filter
- ✓ Paso 4: Configurando el skin del proyecto
- ✓ Paso 5: Revisando el faces-config.xml
- ✓ Paso 6: Revisando las plantillas de la aplicación
- ✓ Paso 7: Listando las cuentas
- ✓ Paso 8: Mostrando detalles de las cuentas
- ✓ Paso 9: Guardando y actualizando cuentas
- ✓ Paso 10: Eliminando cuentas

Reward Dining

Lo esencial de RichFaces

Introducción

- En este laboratorio implementarás controladores básicos para invocar funcionalidades de la aplicación y visualizar resultados para el usuario usando RichFaces.

¿Qué aprenderemos?

- ✓ ¿Cómo construir la infraestructura que requiere una aplicación con RichFaces, sin realizar muchas modificaciones a una arquitectura definida con JSF?
- ✓ ¿Cómo usar etiquetas de RichFaces para construir componentes e interfaces amigables para el usuario?

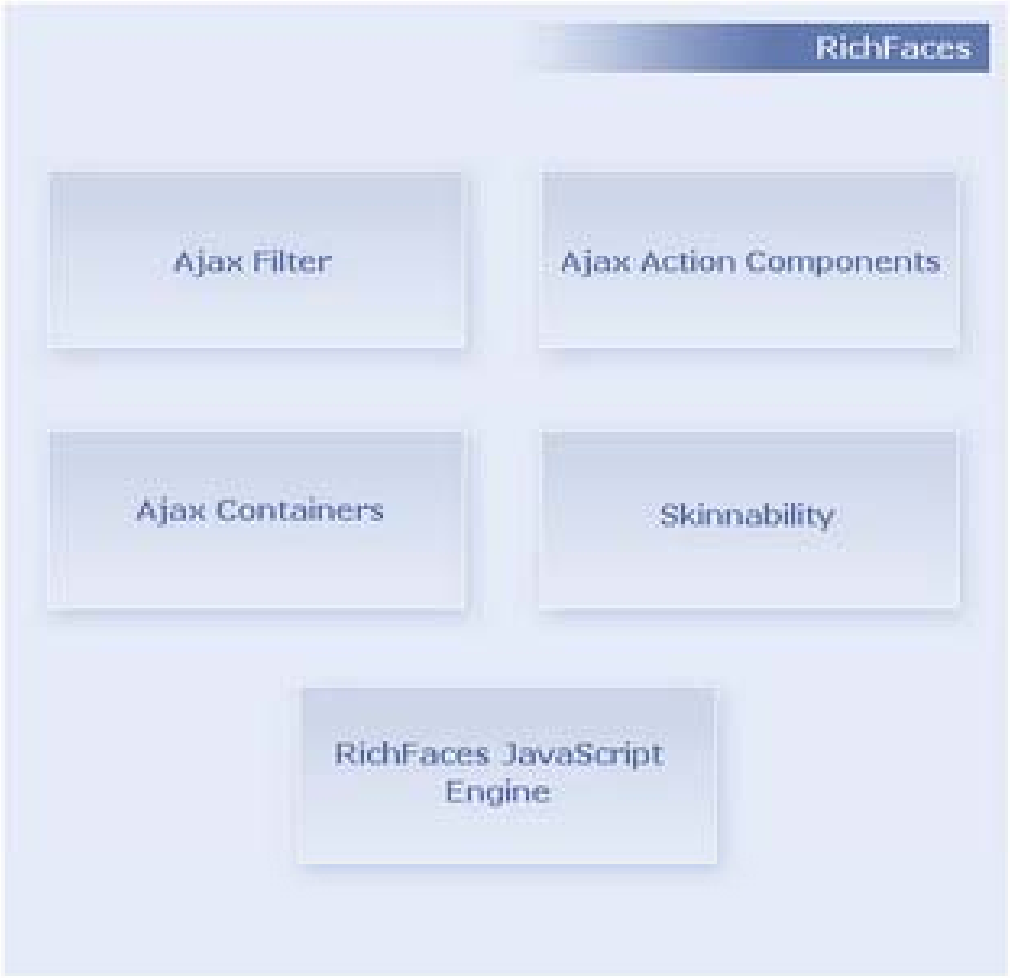
Ganaremos experiencia en…

- ✓ org.ajax4jsf.Filter
- ✓ <navigation-rule/>
- ✓ org.richfaces.SKIN

Introducción a RichFaces

- ✓ No es una implementación de JSF.
- ✓ Es una biblioteca de componentes UI que pueden trabajar con cualquier implementación JSF (1.1, 1.2, 2.0).
- ✓ A la vez un framework muy avanzado para integrarse con AJAX de manera rápida y sencilla.
- ✓ Trabajan con cualquier tecnología de vista: facelets, JSP.
- ✓ Se integra con otros componentes de terceros como: myfaces, Tomahawk, Trinidad, etc.
- ✓ Es un framework que consiste de tres partes: Componentes AJAX JSF, Skinnability, CDK (Component Development Kit)

Introducción a RichFaces



Características principales

- ✓ Dos librerías para el trabajo: rich y a4j.
- ✓ Skinnability.
- ✓ Kit de componentes para desarrollo.

Configurando la infraestructura de RichFaces

- ✓ Ahora si podemos iniciar con la configuración de la aplicación.

Paso 1: Evaluando el estado inicial de la aplicación web

- Al igual que la demo anterior, la aplicación debe permitir a los usuarios ver un resumen de todas las cuentas del sistema, luego poder ver los detalles acerca de una cada una de las cuentas, en particular de una elegida previamente. Pero con un skin diferentes y más funcionalidades:

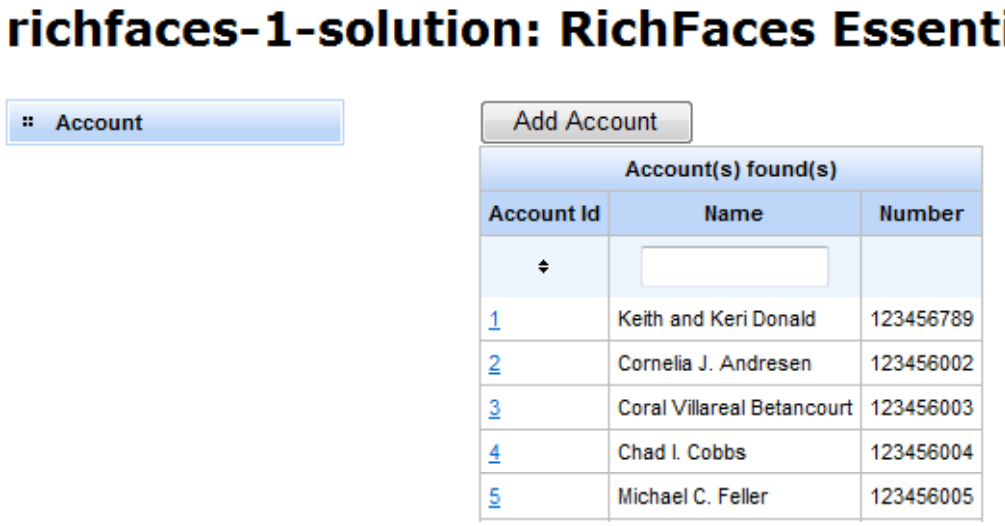


Figura 1: GET pages/accountSummary.jsf: Ver la lista de todas las cuentas y con link en el id de la cuenta para ver los detalles

Paso 1: Evaluando el estado inicial de la aplicación web

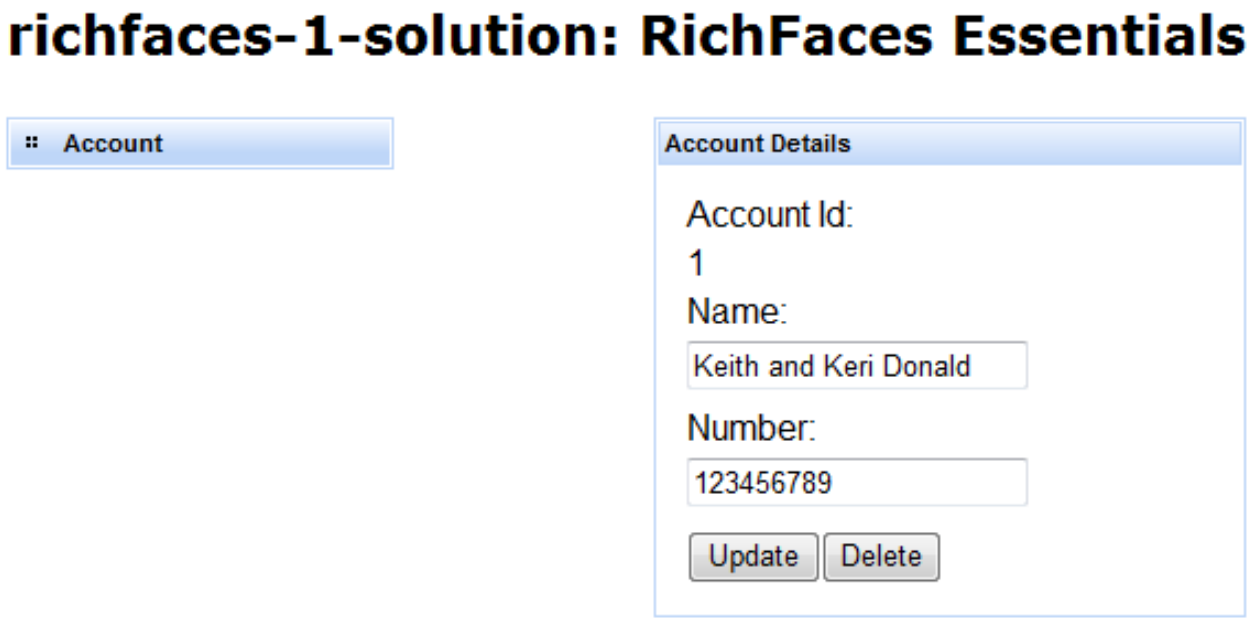


Figura 2: GET /pages/accountDetails.jsf: Ver los detalles de la cuenta con id '1'

Paso 1: Evaluando el estado inicial de la aplicación web

- ✓ Actualmente, esta funcionalidad está implementada a medias. En este primer paso evaluaremos el estado inicial de la aplicación.
- ✓ Empieza por desplegar la aplicación para este proyecto como está. Una vez desplegado, verás que la aplicación muestra un error de tipo 404, lo cual nos indica que el request no ha podido ser procesado. Hay muchas razones por las cuales ha sucedido esto.

Paso 2: Verificando la configuración del backend

- ✓ Rápidamente evalúa la configuración inicial del "backend" de esta aplicación. Para hacer esto, abre el archivo web.xml del directorio src/main/webapp/WEB-INF.

- ✓ Observarás la misma configuración anterior y algunos componentes adicionales.

Paso 3: Configurando el RichFaces Filter

- El elemento principal de la infraestructura central de una aplicación en donde se usa RichFaces es el filter que atiende todos los request de las páginas que usan componentes de RichFaces.

- Verifica en el web.xml y navega hacia la definición RichFaces Filter. Verás que hay un TODO por completar.

```
<filter>
    <display-name>RichFaces Filter</display-name>
    <filter-name>TODO</filter-name>
    <filter-class>org.ajax4jsf.Filter</filter-class>
</filter>
<filter-mapping>
    <filter-name>richfaces</filter-name>
    <servlet-name>Faces Servlet</servlet-name>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
</filter-mapping>
```

Paso 4: Configurando el skin del proyecto

- Otra de las características de RichFaces es el uso de Skins o Themes. La cual nos permite el cambio de estilos y colores de nuestra aplicación tan solo configurando nuestro skin elegido.

- En el archivo web.xml revisa el context param referido al skin usado.

```
<context-param>
    <param-name>org.richfaces.SKIN</param-name>
    <param-value>blueSky</param-value>
</context-param>
```

Paso 5: Revisando el faces-config.xml

- El archivo faces-config.xml continua teniendo la misma configuración que usamos anteriormente.
- Más adelante agregaremos alguna regla adicional de navegación pero por ahora solo dejemos el archivo tal cual.

Paso 6: Revisando las plantillas de la aplicación

- ✓ Navega hacia la carpeta webapp/pages/template y revisa el contenido.
- ✓ Si observas el archivo menu.xhtml podrás notar que ahora el menú de la aplicación ya está usando componentes de RichFaces para construir las opciones de ingreso a las diversas funcionalidades de la aplicación.

```
<rich:panelMenu width="180px">
<rich:panelMenuGroup label="Account">
    <rich:panelMenuItem >
            <h:outputLink value="accountSummary.jsf" >Account Summary</h:outputLink>
    </rich:panelMenuItem>
</rich:panelMenuGroup>
</rich:panelMenu>
```

```
xmlns:rich="http://richfaces.org/rich">
```

Paso 7: Listando las cuentas

- Si revisamos nuestro AccountController, podremos verificar que sigue tal cual lo dejamos.
- Sin embargo, ahora debemos agregar algunas funcionalidades, después de completar el listado y la visualización de los datos de las cuentas.
- Debemos revisar el archivo accountSummary.xhtml, para completar algunos tags que están haciendo falta para renderizar el listado de cuentas.
- Una vez terminados los cambios, procede a levantar la aplicación.
- Si encuentras errores, consulta con tu instructor que puede suceder.
- Si todo salió correcto, continua con el siguiente paso.

Paso 8: Mostrando detalles de las cuentas

- ✓ Ahora que nuestra app ya funciona y se encuentra totalmente configurada, procedemos a implementar la segunda funcionalidad requerida.
- ✓ En nuestra clase AccountController ya contamos con el método que nos permitirá recuperar la información de una cuenta a través de su id.
- ✓ Sin embargo, el botón agregado en el listado, también debe permitirnos agregar cuentas nuevas. Para ello debemos modificar el método accountDetails.
- ✓ Ahora procede a revisar y completar la vista accountDetails.xhtml.

Paso 8: Mostrando detalles de las cuentas

- ✓ Como habrás podido observar hay nuevos componentes:

```
<rich:ajaxValidator event="onkeyup"/>
```

- ✓ Dicho componente se relaciona directamente con las anotaciones de validaciones de Hibernate:

```
@NotEmpty
private String number;
```

- ✓ Además, usaremos un hidden para ocultar el id de la cuenta a la vista del usuario.

```
<h:inputHidden value="#{account.account.entityId}" />
```

- ✓ Por el momento, saquemos el h:panelGroup que contiene las opciones de actualizar/guardar y eliminar, para programarlas después.

- ✓ Ahora inicia la aplicación y verifica si todo está correcto.

Paso 9: Guardando y actualizando cuentas

- Ahora agregaremos dos funcionalidades adicionales: guardar y eliminar las cuentas del sistema.
- Para ello debemos agregar un método que nos permita guardar los datos que podemos modificar o actualizar en nuestra vista accountDetails.xhtml.

```
public String updateAccount(){
    if(account.getEntityId() == null ||
        StringUtils.isBlank(account.getEntityId().toString()) ||
        account.getEntityId().toString().equalsIgnoreCase("0")){
        account.setEntityId(null);
    }
    accountManager.update(account);
    return "accountSummary";
}
```

Paso 9: Guardando y actualizando cuentas

- Al ver el valor de retorno del método updateAccount, nos imaginamos que en realidad esa cadena debe ser la nueva vista a renderizar después de guardar la cuenta.

- Para que sea reconocida la cadena “accountSummary” debe ser registrada en el archivo faces-config.xml como una nueva regla de navegación:

```
<navigation-rule>
    <from-view-id>/*</from-view-id>
    <navigation-case>
        <from-outcome>accountSummary</from-outcome>
        <to-view-id>/pages/accountSummary.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

Paso 9: Guardando y actualizando cuentas

- ✓ Ahora debemos regresamos el h:panelGroup que retiramos anteriormente.
- ✓ Agregamos la acción referenciada al método updateAccount:

```
<h:panelGroup>
    <a4j:commandButton  action="#{account.updateAccount}" value="Update"/>
</h:panelGroup>
```

- ✓ Ahora volvemos a iniciar la aplicación y verificamos el comportamiento de nuestra nueva implementación.

Paso 10: Eliminando cuentas

- Ahora debemos seguir los mismos pasos para implementar la eliminación de una determinada cuenta.

```
public String deleteAccount(){
    accountManager.delete(account.getEntityId());
    return "accountSummary";
}
```

- Pero se debe hacer una validación antes, y es que solo se pueden eliminar cuentas previamente grabadas:

```
<c:if test="${not empty account.account.entityId}">
    <a4j:commandButton action="#{account.deleteAccount}" value="Delete"/>
</c:if>
```

- Ahora debemos probar la nueva implementación. Levanta la aplicación si todo está correcto, has terminado el lab!

Enjoy it!