Java Server Faces (JSF)

Aplicaciones Distribuidas

Curso 2025/2026

Introducción a Java Server Faces

- JavaServer Faces (JSF) es una tecnología del lado del servidor para el desarrollo de aplicaciones web en el entorno Java.
- Proporciona un marco de trabajo que simplifica el desarrollo de interfaces de usuario web, facilitando la interacción entre el cliente y el servidor.
- Proporciona una arquitectura de componentes que permite utilizar un conjunto estándar de componentes de interfaz de usuario, usar componentes de terceros y definir nuevos componentes para la construcción de páginas.
- Los componentes están orientados a eventos, así que JSF proporciona la infraestructura para procesar los eventos generados por la interacción del cliente con los componentes.

Introducción a Java Server Faces

- JSF está pensado para aplicaciones que van a ser utilizadas por multitud de usuarios desde distintos dispositvos, por lo que permite mostrar los componentes de diferentes maneras.
- Se ocupa de sincronizar los componentes de interfaz de usuario con objetos Java (*backing beans* o beans de respaldo) que recogen los valores de entrada y dan respuesta a eventos.
- Realiza la validación de las entradas de usuario.
- Gestiona el flujo entre páginas.
- JSF se ejecuta en el servidor dentro de un contenedor de servlets o contenedor de aplicación web de Java como Apache Tomcat, Jetty, Wildfly o Payara.

Objetivo de JSF

- **Simplificación del Desarrollo**:
  - Se centra en la creación de UI mediante componentes y la separación de la lógica de negocio.
  - Facilita la reutilización y organización de código.
- **Integración Fácil**:
  - Bien integrado con otras tecnologías de Java EE (EJB, CDI, JPA).
  - Facilidad para integrarse en sistemas empresariales.
- JSF sigue siendo popular en aplicaciones empresariales que requieren **robustez** y **mantenibilidad**.
- Está alineado con el ecosistema Java y soportado por Jakarta EE.

Breve historia de JSF

- **2001 JSF 1.0** Publicada la especificación inicial.
- **2006 JSF 1.2** Multitud de mejoras. Compatibildiad hacia atrás, integración con JSP con un lenguaje de expresión unificado. Se incorpora a la plataforma Jave EE 5.
- **2009 JSF 2.0** *Major release*. Facilita la navegación, uso de anotaciones, añade soporte Ajax. Facelets pasa a ser el lenguaje de declaración de vistas por defecto.
- **2013 JSF 2.2** Introduce nuevos conceptos como vistas sin estado, flujo de páginas, injección de artefactos y el uso de CDI.
- **2017 JSF 2.3** Mejora la integración con CDI, soporte a WebSockes.
- **2020 JSF 3.0** Nombre de paquetes pasan de Javax a Jakarta.
- **2022 JSF 4.0** Eliminación de *deprecated*, añade nuevos ámbitos.

Características Clave de JSF

1. **Componentes Reutilizables**: Componentes que encapsulan lógica de presentación (formulario, botón, panel, etc.).
2. **Navegación Declarativa**: Define el flujo de la aplicación sin necesidad de programación explícita.
3. **Ciclo de Vida de Solicitud**: Control de flujo estructurado desde la solicitud hasta la respuesta.
4. **Soporte para AJAX**: Facilita la creación de aplicaciones interactivas sin recargar la página completa.
5. **Facelets como Motor de Vistas**: Mejora sobre JSP, con una sintaxis XML más limpia y modularidad en las plantillas.

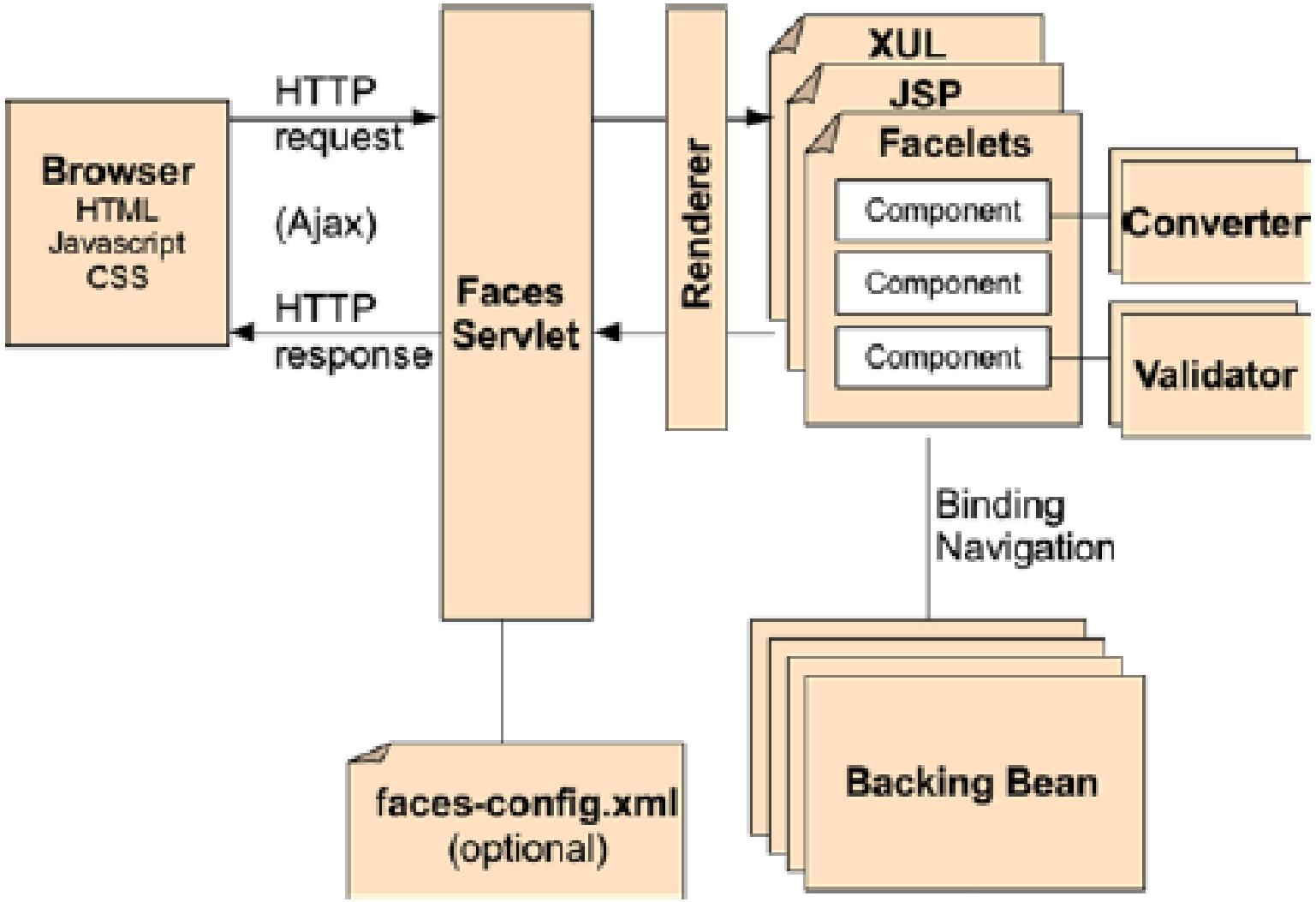
Context Dependency Injection (CDI)

- Inyección de dependencias: desacopla objetos dependientes. Los objetos reciben sus dependencias en lugar de crearlas ellos mismos. Esto permite desacoplar las clases.
- En JEE, es el contenedor quién gestiona el ciclo de vida de los componentes y realiza la inyección de los mismos.
- CDI, incluido a partir de Java EE 6, proporciona un API para gestionar el contexto del ciclo de vida de los componentes e inyectarlos de forma segura.
- Desde la versión 2.3, JSF lo usa ampliamente.
- Etiquetas `@Named` y `@Inject`.

Patrón Modelo-Vista-Controlador

- JSF utiliza una variación del patrón Modelo-Vista-Controlador (MVC) específica para aplicaciones web.
- Este patrón surge de la necesidad de separar en las aplicaciones la lógica de negocio del código para la interfaz de usuario.
- De esta forma tendremos un **Modelo** con la lógica de negocio y datos, la **Vista**, que será la capa de presentación y el **Controlador** que será código de aplicación que responde a eventos de usuario e integra el modelo y la vista.
- Hace el código más escalable y reduce dependencias entre las distintas capas.
- En JSF el controlador es un Servlet.

Arquitectura de JSF



Arquitectura de JSF: Elementos

- **FacesServlet**: Servlet principal que actúa de controlador en JSF. Se puede configurar opcionalmente con el fichero `faces-config.xml`
- **Páginas y componentes**:
  - JSF puede utilizar cualquier lenguaje de declaración de páginas, se recomienda utilizar **Facelets**.
  - Facelets está formado por un árbol de componentes que proporcionan funcionalidad específica para interactuar con el usuario final.
- **Renderizador**: responsable de mostrar un componente y traducir la entrada del usuario a los valores de las propiedades del componente.

Arquitectura de JSF: Elementos

- **Conversores y Validadores**:
  - Los conversores convierten el valor de un componente (Integer, Date, Boolean) a o desde valores de lenguaje de marcado (String) para ser mostrado.
  - Los validadores se encargan de asegurarse que los valores introducidos por el usuario son válidos.
- **Backing beans (beans de respaldo)**: clase java que sincroniza valores con componentes, procesa lógica de negocio y gestiona la navegación entre páginas.
- **Soporte AJAX**: desde la versión 2.2, JSF soporta AJAX.
- **Lenguaje de expresión**: para hacer mapeo en las páginas JSF de variables y acciones entre los componentes y los beans de respaldo.

FacesServlet

- Cuando ocurre un evento, la notificación del evento se envía por HTTP al servidor y es interceptada por **FacesServlet**.
- **FacesServlet** le pasa el control al objeto **LifeCycle** y crea el objeto **FacesContext** por medio de una factoria.
- El objeto **LifeCycle** utiliza el objeto **FacesContext** para procesar la petición de entrada y crear la correspondiente respuesta.
- **FacesServlet** tiene una configuración por defecto, pero se puede configurar en `web.xml`.
- Desde el punto de vista del framework JSF, FacesServlet hace de controlador en su versión de MVC.

FacesContext

- Representa toda la información de contexto asociada con el procesamiento de una petición de entrada y la creación de una respuesta.
- Permite la interacción con la interfaz de usuario y el resto del entorno JSF.
- Incluye métodos para añadir mensajes en la interfaz u obtener objetos asociados con el contexto de la aplicación.

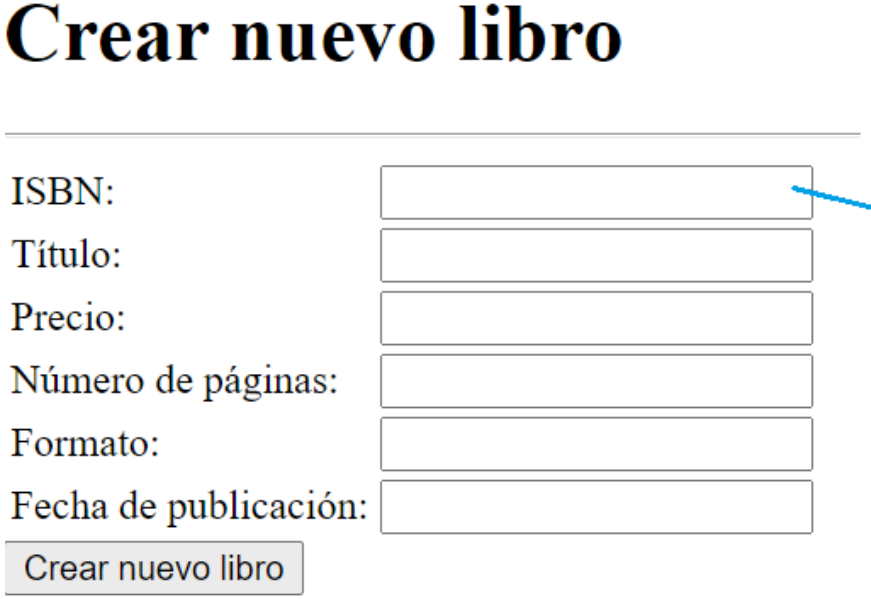
Backing Beans

- Los beans de respaldo (**Backing Beans**) también llamados beans administrados (**Managed Beans**) son la puerta de acceso al modelo en el patrón MVC.
- Clases java anotadas.
- Puedes implementar lógica de negocio o delegarla en otras clases (servicios, EJBs, etc.).
- Manejan navegación entre páginas.
- Los datos se mantienen dentro de los atributos del backing bean y las acciones de una página activan alguno de sus métodos.
- Para vincular componentes a un backing bean, se utiliza lenguaje de expresión de JSF.

Lenguaje de expresión de JSF

- Se utiliza para vincular las páginas JSF con los backing bean.
- La sintaxis básica para una sentencia es: `#{expr}`.
- Las sentencias de Lenguaje de Expresión se parsean y evalúan por JSF.
- Se puede utilizar para sentencias lógicas y matemáticas y también es posible mezclar valores literales con expresiones.
- Permite utilizar la mayoría de operadores java:
  - Aritméticos: `+`, `-`, `*`, `/ (div)`, `% (mod)`
  - De comparación: `== (eq)`, `!= (ne)`, `< (lt)`, `> (gt)`, `<= (le)`, `>= (ge)`.
  - Lógicos: `&& (and)`, `|| (or)`, `! (not)`

Lenguaje de expresión de JSF




```
<h1>Crear nuevo libro</h1>
<hr />
<h:messages id="errors" infoStyle="color:blue" warnStyle="color
    errorStyle="color:red" showDetail="true"/>
<h:form id="nuevolibro">
    <h:panelGrid columns="2">
        <h:outputLabel value="ISBN: " />
        <h:inputText value="#{libroConstructor.isbn}" />
        <h:outputLabel value="Titulo:" />
        <h:inputText value="#{libroConstructor.titulo}" />
        <h:outputLabel value="Precio: " />
        <h:inputText value="#{libroConstructor.precio}" />
        <h:outputLabel value="Número de páginas: " />
        <h:inputText value="#{libroConstructor.numPaginas}" />
        <h:outputLabel value="Formato: " />
        <h:inputText value="#{libroConstructor.formato}" />
        <h:outputLabel value="Fecha de publicación: " />
        <h:inputText value="#{libroConstructor.fecha}">
            <f:convertDateTime pattern="dd-MM-yyyy" />
        </h:inputText>
    </h:panelGrid>
    <h:commandButton value="Crear nuevo libro"
        action="#{libroConstructor.crearLibro()}">
        <f:ajax execute="@form" render=":errors :booklist"/>
        </h:commandButton>
```

```
@ManagedBean
@ViewScoped
public class LibroConstructor implements Serializable{

    protected String titulo;
    protected String formato;
    protected Date fecha;
    protected Integer numPaginas;
    protected Float precio;
    protected String isbn;

    public String getIsbn() {
        return isbn;
    }
    public void setIsbn(String isbn) {
        this.isbn = isbn;
```

Lenguaje de expresión de JSF

- El lenguaje de expresión proporciona un conjunto de objetos implícitos que permiten al desarrollador acceder a parámetros de la petición que está siendo procesada o al enterno de ejecución.
  - Objetos como `facesContext`, `request`, `session`, `param` o `resource`.

Ejemplo	Descripción
`#{miBean.value}`	Retorna la propiedad value del objeto almacenado bajo la clave miBean
`#{miBean['value']}`	Es lo mismo que `#{miBean.value}`
`#{miArrayList[5]}`	Retorna el quinto elemento de una lista alamcenada bajo la clave miArrayList.
`Estas #{(user.balance > 100) ? 'logueado' : 'no logueado'}`	Retorna la cadena "Estás logueado" si la propiedad balance del objeto almacen bajo la clave user es mayor que 100. Retorna "Estás no logueado" en otro caso.

Soporte AJAX

- AJAX (Asynchronous JavaScrip and XML): permite que las aplicaciones interactúen con el servidor de manera asíncrona.
- Se envían datos de componentes individuales, se procesan en el servidor y se actualiza la interfaz sin volver a cargarla.
- Mejora la experiencia de usuario al hacer las interacciones con la interfaz más rápidas y fluidas.
- Solo se envían y reciben datos de los componentes que cambian.

Etiqueta `f:ajax`

- Sirve para activar el comportamiento AJAX en componentes JSF.
- Atributos:
  - `event`: define el evento que activará la solicitud AJAX.
  - `render`: especifica los Ids de los componentes que deben actualizarse tras la respueta AJAX.
  - `execute`: Ids de los componentes que se envían al servidor para ser procesados. Por defecto se envía solo el actual.
  - `listener`: asocia un método del bean de respaldo que se ejecutará en el servidor.
- No es necesaria en librerías de componentes basados en AJAX como **Primefaces**.

Definiendo Backing Beans

- Son gestionados por el Faces Servlet y requieren cumplir:
  - La clase debe ser anotada:
    - A partir de JSF 2.3 con la anotación CDI (Context dependency Injection) `@Named (javax.inject.Named)`. Esto hace que estén inmediatamente disponibles para el contexto del Lenguaje de Expresión y por lo tanto, para las vistas.
  - Debe tener una anotación de ámbito.
  - La clase debe ser pública y no puede ser final o abstracta.
  - Debe tener un constructor público sin argumentos.
  - Los atributos que se vinculan a componentes de la vista deben tener métodos `getter` y `setter`.

Ámbitos de los Backing Beans

- Representan el tiempo de vida del *backing bean*, el alcanse de sus atributos y métodos: De mayor a menor tiempo de vida:
- `@javax.enterprise.context.ApplicationScoped` : el bean tiene el mismo tiempo de vida que la aplicación. Se representa como un atributo del ServletContext. Solo existirá una instancia compartida entre todas las peticiones y sesiones. Pueden ser usados para datos que se utilizan por toda la aplicación y que deben ser configurados una única vez.
- `@javax.enterprise.context.SessionScoped` : el tiempo de vida del bean está vinculado al tiempo de vida de la sesión HTTP establecida. Son útiles para controlar datos específicos del cliente, como el usuario conectado, el lenguaje seleccionado u otras preferencias.

Ámbitos de los Backing Beans

- `@javax.enterprise.context.ConversationScoped`: ligado al tiempo de vida de la instancia de `Conversation`. Esta instancia ofrece métodos `begin()` y `end()` a los que hay que invocar de forma específica para indicar comienzo y fin del ámbito. Si no se invocan, el bean se comporta como si fuera de ámbito `@RequestScoped`. Son útiles a la hora de volver a una página con estado durante la misma sesión después de haber sido redirigido a alguna otra página.
- `@javax.faces.flow.FlowScoped`: similar a `@ConversationScoped` solo que se reduce a un conjunto específico de vistas.

Ámbitos de los Backing Beans

- `@javax.faces.view.ViewScoped`: ligado al tiempo de vida del estado de una vista JSF. Es decir, mientras el cliente siga haciendo peticiones en una misma vista, el bean de respaldo seguirá existiendo. Si una petición implica navegar a otra vista, el bean dejará de existir.
- `@javax.enterprise.context.RequestScoped`: ligado al tiempo de vida de la petición HTTP. Útiles para formularios estáticos que no tienen actualizaciones con Ajax.
- `@javax.enterprise.context.Dependent`: ligado al tiempo de vida del ámbito en el que está siendo creado. Si por ejemplo se inyectan en un bean con ámbito de aplicación, tendrán ese mismo ámbito.

Ámbitos de los Backing Beans

- El ámbito que se elige depende solamente de los datos y el estado que representa el bean. Lo lógico es empezar por el ámbito más bajo e ir subiendo a mayores ámbitos dependiendo de las necesidades.
- Ya que el ámbito marca cuando los beans son creados y es el contenedor el que va a gestionar todo su ciclo de vida, disponemos de las etiquetas `@PostConstruct` y `@PreDestroy` para anotar métodos que queramos que se ejecuten justo después de crear la instancia del bean o justo antes de destruirla.

Navegación en JSF

- Las aplicaciones web tienen multitud de páginas entre las que hay que navegar. JSF tiene varias opciones de navegación que permiten controlar el flujo entre páginas.
- Se puede navegar fácilmente a otra página sabiendo su nombre.
  - Si sólo se necesita ir de una página a otra, usando un link o botón: componentes `<h:button>`, `<h:link>`, `<h:outputLink>`.

```
<h:link value ="Crear nuevo libro " outcome ="nuevoLibro.xhtml "/>
```

  - Si se necesita realizar algún procesado antes de navegar, con componentes `<h:commandButton>`, `<h:commandLink>`.

```
<h:commandButton value ="Crear nuevo libro"
action ="#{libroWeb.crearLibro}"/>
```

  - Librerías como Primefaces tienen componentes equivalentes.

Navegación en JSF

- El método invocado en el action puede devolver un String con el nombre de la página a la que navegar.

```
@Named
@RequestScoped
public class LibroWeb implements Serializable {
// ...
public String crearLibro (){
// ...
// realizar cualquier procesamiento
// ...
  return "nuevoLibro.xhtml";
 }
```

Navegación en JSF

- Existe el método `redirect` a partir del objeto `FacesContext` para desplegar una página desde cualquier clase.

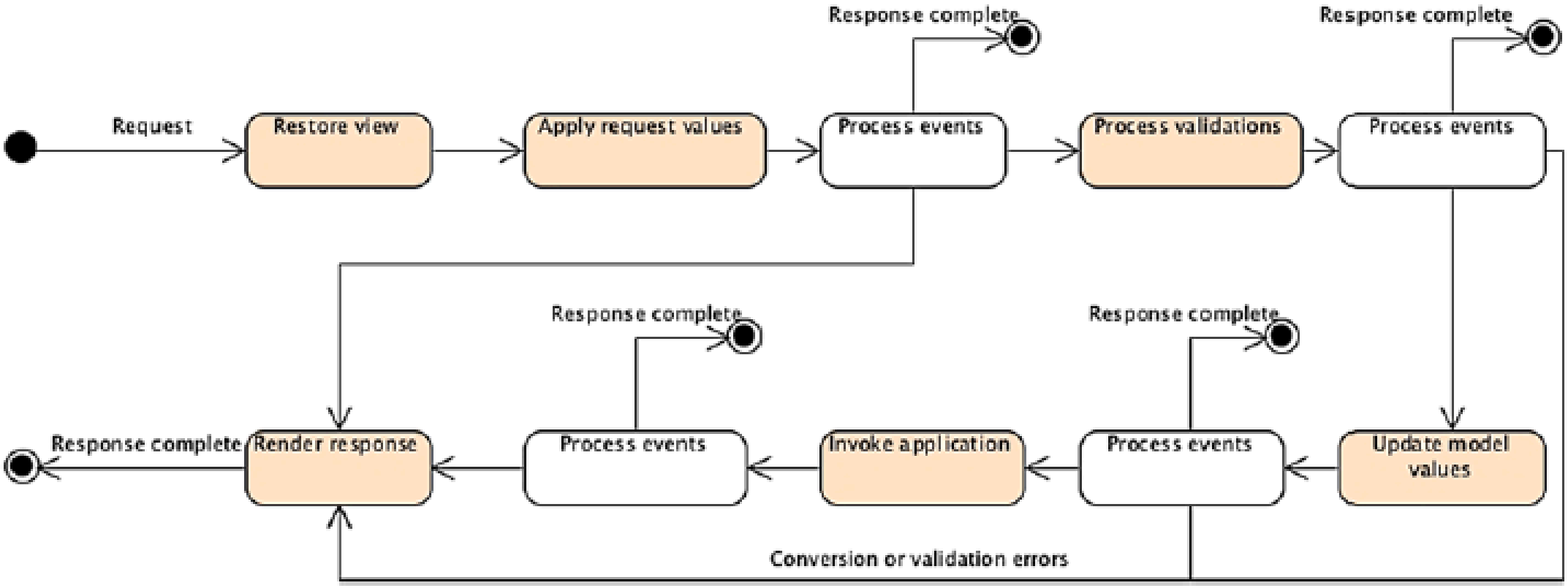
```
facesContext.getExternalContext().redirect("nombreVista");
```

- Si se devuelve una página como resultado de un método invocado en un `action`, se suele añadir el parámetro `?faces-redirect=true"` para que se navegue por redirección.

Páginas JSF

- Una página JSF es un XHTML.
- Un XHTML es un documento que tiene sus raíces en HTML pero está reformulado en estricto XML.
- Sigue un XML schema y tiene una representación gráfica en los navegadores.
- Los XHTML en JSF:
  - Definen un listado de librerías de etiquetas en la cabecera.
  - El cuerpo contiene la representación gráfica de la página.

Página JSF. Ciclo de vida



Página JSF. Ciclo de vida

- JSF parsea la definición de la vista a un árbol de componentes, siendo la raíz del árbol una instancia `view root` asociada a la instancia del objeto `FacesContext`.
- Este árbol tiene un ciclo de vida específico.
- Pulsar un botón causa una petición que se envía desde el navegador hasta el servidor.
- Esta petición se traduce a un evento que pueda ser procesado por la lógica de la aplicación en el servidor.
- Todos los datos de entrada pasan por una fase de validación.
- JSF es responsable de que los componentes gráficos se rendericen apropiadamente.

Fase 1: Restore View (Restaurar Vista)

- **Propósito**: Crear o restaurar el árbol de componentes de la vista actual.
- **Qué Ocurre**:
  - Si es una **nueva solicitud**, se crea un nuevo árbol de componentes (`UIViewRoot`).
  - Si es una **solicitud posterior** (como un formulario enviado), se restaura el árbol existente (`UIViewRoot` previo).
- **Acceso al FacesContext**: Se inicializa el `FacesContext` para almacenar el estado de la solicitud.
- **Resultado**: La vista actual queda lista para procesar la solicitud del usuario.

Fase 2: Apply Request Values (Aplicar Valores de la Solicitud)

- **Propósito**: Procesar y asignar los valores de entrada proporcionados por el usuario a los componentes.
- **Qué Ocurre**:
  - Cada componente UI en el árbol recupera su valor del `FacesContext`.
  - Estos valores suelen provenir de campos de formulario enviados por el cliente.
  - **Manejo de Errores**: Si un valor es inválido, se registra un mensaje de error en el `FacesContext`.
- **Resultado**: Los valores están disponibles, pero aún no han sido validados ni aplicados al bean de respaldo.

Fase 3: Process Validations (Procesar Validaciones)

- **Propósito**: Validar los valores introducidos según reglas definidas en los componentes.
- **Qué Ocurre**:
  - Se verifican reglas de validación y conversión (como formato de email, rango numérico).
  - Si un valor no pasa la validación, se registra un error y JSF salta a la fase `Render Response` que muestra los mensajes de error.
- **Resultado**: Si los valores son válidos, continua el ciclo.

Fase 4: Update Model Values (Actualizar Valores del Modelo)

- **Propósito**: Transferir los valores validados desde los componentes a los Beans de respaldo.
- **Qué Ocurre**:
  - Los valores se asignan a las propiedades correspondientes de los Beans de respaldo.
  - **Errores**: Si ocurre un error en la asignación de valores, se registra en el `FacesContext`.
- **Resultado**: Los datos introducidos por el usuario están sincronizados con el modelo de presentación.

Fase 5: Invoke Application (Invocar Aplicación)

- **Propósito**: Ejecutar la lógica de negocio en respuesta a la acción del usuario.
- **Qué Ocurre**:
  - Se ejecuta el método de acción vinculado a la solicitud (e.g., un botón de enviar).
  - Se determina el resultado de la navegación (la vista siguiente).
  - **Navegación**: Dependiendo del resultado, JSF determina si redirige a otra página o recarga la vista.
- **Resultado**: Se ha completado la lógica de negocio y la aplicación está lista para renderizar la respuesta.

Fase 6: Render Response (Renderizar Respuesta)

- **Propósito**: Generar la respuesta final para el cliente.
- **Qué Ocurre**:
  - JSF construye la respuesta HTML usando el árbol de componentes UI.
  - Si hay errores, la página actual se vuelve a renderizar mostrando los mensajes de error.
- **Resultado**: La respuesta se envía al cliente y finaliza el ciclo de vida de la solicitud.

Ventajas y desventajas de JSF

Ventajas

- Reduce el esfuerzo de crear y mantener aplicaciones.
- Proporciona componentes de interfaz de usuario reutilizables.
- Facilita gestionar los estados de la interfaz entre varias peticiones al servidor.
- Facilita transferir datos entre los componentes UI.
- Abstracción del ciclo de vida de una solicutd HTTP.
- Su gestión de eventos es muy robusta.
- Permite implementar componentes propios.
- Dentro de la especificación JEE.

Ventajas y desventajas de JSF

Desventajas

- Se considera que tiene una curva de aprendizaje pronunciada.
- Genera un código HTML complejo.
- Dependencia del contenedor JEE.
- Ha ido perdiendo popularidad en los últimos años frente a frameworks JavaScript, arquitecturas basadas en microservicios y otros framework Java como Spring MVC.