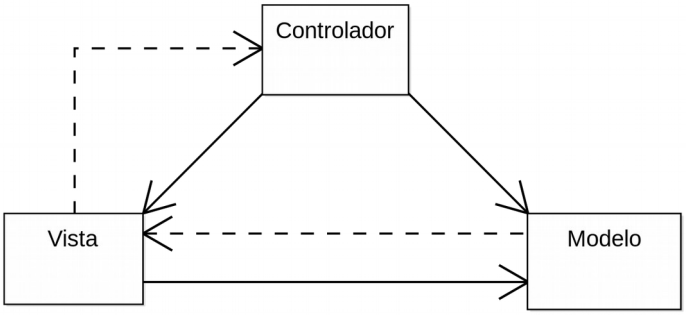
Sumario

Java Server Faces JSF

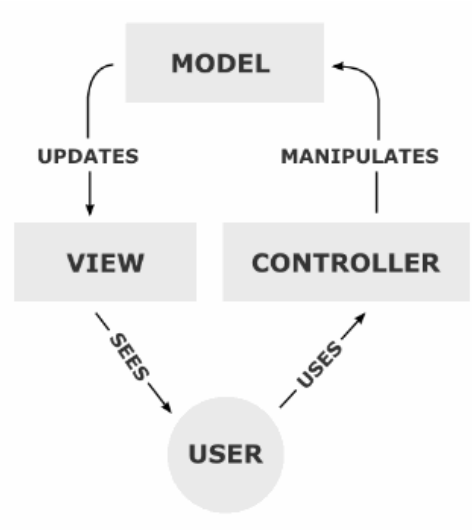
Modelo Vista Controlador (MVC)

Modelo-vista-controlador (**MVC**) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el **modelo**, la **vista** y el **controlador**, es decir, por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.



De manera genérica, los componentes de MVC se podrían definir como sigue:

- El **Modelo**: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
- El **Controlador**: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo' (por ejemplo, desplazamiento o scroll por un documento o por los diferentes registros de una base de datos), por tanto se podría decir que el 'controlador' hace de intermediario entre la 'vista' y el 'modelo' (véase ***Middleware***).
- La **Vista**: Presenta el 'modelo' (información y *lógica de negocio*) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.



MVC y Bases de Datos

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos el cual gestiona los datos que debe utilizar la aplicación; en líneas generales del **MVC** dicha gestión corresponde al modelo. La unión entre *capa de presentación* y *capa de negocio* conocido en el paradigma de la Programación por capas representaría la integración entre la **Vista** y su correspondiente **Controlador** de eventos y acceso a datos, MVC no pretende discriminar entre capa de negocio y capa de presentación pero si pretende separar la capa *visual gráfica* de su correspondiente *programación y acceso a datos*, algo que mejora el desarrollo y mantenimiento de la *Vista* y el *Controlador* en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

Uso en aplicaciones Web

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitud de frameworks, comerciales y no comerciales, que implementan este patrón (ver apartado siguiente "*Frameworks MVC*"); estos frameworks se diferencian básicamente en la interpretación de como las funciones MVC se dividen entre cliente y servidor.

Los primeros frameworks MVC para desarrollo web planteaban un enfoque de cliente ligero en el que casi todas las funciones, tanto de la vista, el modelo y el controlador recaían en el servidor. En este enfoque, el cliente manda la petición de cualquier hiperenlace o formulario al controlador y después recibe de la vista una página completa y actualizada (u otro documento); tanto el modelo como el controlador (y buena parte de la vista) están completamente alojados en el servidor. Como las tecnologías web han madurado, ahora existen frameworks como JavaScriptMVC, Backbone o jQuery que permiten que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente (véase AJAX).

Para el desarrollo de aplicaciones empresariales se utiliza frecuentemente el patrón de diseño MVC Modelo Vista Controlador que además es sencillo de implementar en el desarrollo web. En este patrón el modelo es modificable por las funciones de negocio. Estas funciones son solicitadas por el

usuario mediante el uso de un conjunto de vistas de la aplicación que solicitan dichas funciones de negocio a través de un controlador, que es el módulo que recibe la peticiones de las vistas y las procesa.
Se suele clasificar en dos tipos a las aplicaciones basadas en MVC:

● Las vistas conocen la acción que se va a invocar en su petición.
● El controlador introduce un conjunto de reglas que mapean a las peticiones con las funciones, controlando además el flujo de navegación por la aplicación.

Un ejemplo de aplicaciones de tipo 1 son las que se construyen utilizando JSF y como ejemplo de tipo 2 serian las creadas con Spring MVC.

La creación de aplicaciones basadas en el patrón MVC se ve facilitada por el uso de frameworks. Un frameworks es un conjunto de APIs y módulos normalmente acompañados de la documentación y guía de uso que definen la manera de implementar alguna de las capas de nuestra aplicación.

Que es Java Server Faces (JSF)?

JSF es un framework para crear aplicaciones J2EE basadas en el patrón MVC de tipo 1. JSF tiene como características principales:

● Utiliza páginas JSP para generar las vistas, añadiendo una biblioteca de etiquetas propia para crear los elementos de los formularios

● HTML:
  ➔ Asocia a cada vista con formularios a un conjunto de objetos java manejados por el controlador (managed beans) que facilitan la recogida, manipulación y visualización de los valores mostrados en los diferentes elementos de los formularios.

  ➔ Introduce una serie de etapas en el procesamiento de la petición, como por ejemplo la de validación, reconstrucción de la vista, recuperación de los valores de los elementos, etc.

  ➔ Utiliza un sencillo archivo de configuración para el controlador en formato xml.

  ➔ Es extensible, pudiendo crearse nuevos elementos de la interfaz o modificar los ya existentes.

  ➔ Forma parte del estándar J2EE. En efecto, hay muchas alternativas para crear la capa de presentación y control de una aplicación web java, como Struts y otros frameworks, pero solo JSP forma parte del estándar.

JSF nos permite desarrollar rápidamente aplicaciones de negocio dinámicas en las que toda la lógica de negocio se implementa en java, o es llamada desde java, creando páginas para las vistas muy sencillas (salvo que introduzcamos mucha maquetación HTML o Javascript)

Ventajas

● El código JSF con el que creamos las vistas (etiquetas jsp) es muy parecido al HTML estándar. Lo pueden utilizar fácilmente desarrolladores y diseñadores web.

● JSF se integra dentro de la página JSP y se encarga de la recogida y generación de los valores de los elementos de la página.

● JSF resuelve validaciones, conversiones, mensajes de error e internacionalización (i18n)

● JSF permite introducir javascript en la página, para acelerar la respuesta de la interfaz en el cliente (navegador del usuario).

● JSF es extensible, por lo que se pueden desarrollar nuevos componentes a medida, También se puede modificar el comportamiento del framework mediante APIs que controlan su funcionamiento.

● JSF forma parte del estándar J2EE, mientras que otras tecnologías para creación de vistas de las aplicaciones no lo forman.

● JSF dispone de varias implementaciones diferentes, incluyendo un conjunto de etiquetas y APIs estándar que forman el núcleo del framework. Entre estas implementaciones cabe destacar la implementación de referencia de Oracle, actualmente desarrollada como un proyecto open source, y la implementacióndel proyecto Apache, MyFaces, dotada de un conjunto de extensiones que la hacen muy interesante para el desarrollo de aplicaciones corporativas.

Cómo funciona JSF?

Normalmente las aplicaciones web se construyen como un conjunto de pantallas con las que va interactuando el usuario. Estas pantallas contienen textos, botones, imágenes, tablas y elementos de selección que el usuario modifica. Todos estos elementos están agrupados en formularios HTML, que es la manera en que las páginas web envían la información introducida por el usuario al servidor.
La principal función del controlador JSF es asociar a las pantallas, clases java que recogen la información introducida y que disponen de métodos que responden a las acciones del usuario. JSF nos resuelve de manera muy sencilla y automática muchas tareas:

● Mostrar datos al usuario en campos de texto y tablas.

● Recoger los datos introducidos por el usuario en los campos del formulario.

● Controlar el estado de los controles del formulario según el estado de la aplicación, activando, ocultando o añadiendo y eliminando controles y demás elementos.

● Realiza validaciones y conversiones de los datos introducidos por el usuario.

● Llena campos, listas, combos y otros elementos a medida que el usuario va interactuando con la pantalla.

● Controla los eventos que ocurren en los controles (pulsaciones de teclas, botones y movimientos del ratón).

Las aplicaciones JSF están formadas por los siguientes elementos principales:

● Páginas JSP que incluyen los formularios JSF. Estas páginas generarán las vistas de la aplicación.

● Beans java que se conectan con los formularios JSF.

● Clases java para la lógica de negocio y utilidades.

● Archivos de configuración, componentes a medida y otros elementos del framework.

● Resto de recursos de la aplicación web: recursos estáticos, javascript y otros elementos.

Las etiquetas JSF

JSf dispone de un conjunto básico de etiquetas que permiten crear fácilmente componentes dinámicos en las páginas web. Estas etiquetas son:

● **h:commandButton.** Un botón al que podemos asociar una acción.

● **h:commandLink.** Un enlace hipertexto al que podemos asociar una acción.

● **h:dataTable.** Crea una tabla de datos dinámica con los elementos de una propiedad de tipo Array o Map del bean.

● **h:form.** Define el formulario JSF en la página JSP-

● **h:graphicImage.** Muestra una imagen jpg o similar.● h:inputHidden. Incluye un campo oculto del formulario.

● **h:inputSecret.** Incluye un campo editable de tipo contraseña (no muestra lo que se escribe)

● **h:inputText.** Incluye un campo de texto normal.

● **h:inputTextarea.** Incluye un campo de texto multilínea.

● **h:message.** Imprime un mensaje de error en la página (si se ha producido alguno).

● **h:messages.** Imprime varios mensajes de error en la página, si se han producido.

● **h:outputFormat.** Muestra texto parametrizado. Utiliza la clase java.text.MessageFormat de formateo.

● **h:outputLabel.** Muestra un texto fijo.

● **h:outputLink.** Crea un enlace hipertexto.

● **h:panelGrid.** Crea una tabla con los componentes incluidos en el panelGrid.

● **h:panelGroup.** Agrupa varios componentes para que cierto componente los trate como un único componente (por ejemplo para meter varios componentes en una celda de un panelGrid.

● **h:selectBooleanCheckbox.** Crea una casilla con dos estados: activado y desactivado.

● **h:selectManyCheckbox.** Crea un conjunto de casillas activables.

● **h:selectManyListbox.** Crea una lista que permite seleccionar múltiples elementos.

● **h:selectManyMenu.** Crea una lista desplegable de selección múltiple.

● **h:selectOneListbox.** Crea una lista en la que se puede seleccionar un único elemento.

● **h:selectOneMenu.** Crea una lista desplegable de selección.

● **h:selectOneRadio.** Crea una lista de botones, redondos normalmente, excluyentes.Backbeans

A las clases java que se asocian a los formularios JSF se les denomina backend beans ya que son los beans (clases java) que están detrás del formulario.
Estos beans se referencian en el archivo o con anotaciones de configuración de JSF en el apartado de managed beans, ya que son beans gestionados por el controlador JSF. Este se encarga de su construcción y destrucción automáticas cuando es necesario.

Estructura de las páginas

En su versión más sencilla, cada página JSF está formada por una página JSP que contiene un formulario (HTML FORM) y un backbean. El controlador JSF registra en el servidor de aplicaciones un tipo especial de petición, típicamente *.jsf, que estará asociado a estas páginas.
El primer caso comienza cuando el usuario realiza en su navegador una petición a una url de tipo *.jsf. Cuando al servidor web llega una petición del tipo JSF, el controlador JSF entra en funcionamiento. Primero comprueba si es la primera vez que se accede a dicha página. Si es así, carga la página jsp asociada página jsp y la procesa construyendo en memoria la representación de los controles de la página. Tras esta etapa JSF sabe cómo construir el código HTML de salida y la lista de controles de usuario que la cumplen, es decir, sabe lo que contiene y cómo pintarla.
El siguiente paso es asociar los backbeans. Para ello, del procesamiento de la página jsp, el controlador ha obtenido la lista de backbeans asociados, por lo que procede a buscarlos en sus correspondientes ámbitos de la aplicación como la request y la session. Los beans que no existan se crean llamando a los constructores de sus clases, definidos en la sección de managed beans del archivo de configuración de JSF. El tercer paso es dar valores a las propiedades de los elementos JSF de la página. Aquí juega un papel fundamental el lenguaje de expresiones de JSF, que es parecido al lenguaje de expresiones que se permite en las páginas jsp normales. En su versión más sencilla una expresión JSF sería del tipo **#{mibackbean.propiedad}.**
Finalmente el servidor devuelve al usuario una página creada a partir de una página JSP que incluye normalmente etiquetas JSF, cuyos valores se extraerán del backbean asociado, ahora ya actualizados.

Respondiendo a las acciones del usuario

Una vez que el usuario ve la página web que se ha construido con JSF, está listo para comenzar a interactuar con ella. El método más sencillo de interacción es el uso de formularios web.

Un formulario web simple consta de:

● Etiquetas que muestran información.
● Campos editables.
● El botón de envío del formulario.

El controlador JSF dispone de un conjunto de etiquetas que permiten definir formularios JSF. Las más sencillas son:

● **h:form.** Esta etiqueta sustituye al form de HTML, añadiendo la funcionalidad JSF al formulario

● **h:outputText.** Sirve para imprimir valores en la página.

● **h:inputText.** Sirve para crear campos editables en los que introducir los datos.

● **h:commandButton.** Crea botones que envían el formulario.

Cuando la página JSF contiene elementos que incluyen acciones se ejecuta una fase más en el procesamiento de la petición al servidor. Si en nuestro formulario hay botones u otros elementos que tienen una propiedad action, si se pulsa sobre el elemento cuando la petición sea procesada por el servidor se ejecutará la lógica de la acción asociada a este elemento. Este es el mecanismo habitual de JSF para ejecutar la lógica de la aplicación. Esto se hace ejecutando los métodos del backbean asociado a la página.

La navegación entre páginas

Cuando se ejecuta una petición que incluye una acción se ejecuta el mecanismo de navegación de JSF. Tras la ejecución de la acción, el controlador determina cómo se debe mostrar al usuario el resultado de la petición. Hay varias posibilidades:

● Finalizar la petición mostrando la página jsp que originó la petición, que es la opción por defecto.

● Mostrando otra página jsp diferente.

● Enviando al usuario una petición de redirección, por lo que el navegador del usuario se dirigirá automáticamente a otra página cuando reciba la respuesta a su petición.

Este mecanismo de navegación se implementa de manera sencilla en la página JSF. Cuando el controlador JSF llama al método asociado a la acción, este devuelve un valor de tipo String. Este valor es utilizado junto con las reglas de navegación creadas en el archivo de configuración de JSF para determinar la página que se debe enviar como respuesta al usuario.Las reglas de navegación definen:

● La página de origen. Indica el jsp que originó la petición.

● La etiqueta de destino. Es la cadena que identifica el destino. Esta cadena es devuelta por el método del backbean que procesa la acción.

● La página de destino para cada etiqueta. Normalmente es el jsp que procesa la petición de salida, utilizando los datos que hay en la request y en la sesión.

● Si es un envío directo interno o una redirección externa. En el primer caso la respuesta se generará en la misma petición mediante una redirección interna a otro jsp o servlet. En el segundo caso se enviará como respuesta al navegador una instrucción de redirección para que el navegador realice una nueva petición de otra página.

Además las direcciones de origen admiten el * para que una misma regla sirva para múltiples páginas. También se pueden poner reglas por defecto que se aplican a todas las peticiones.

Las etapas del procesamiento de la petición

Para entender el procesamiento de una página JSF hay que entender el ciclo de vida de la petición dentro del controlador JSF. Este ciclo de vida está compuesto de 6 fases.

Durante el procesamiento de una petición el controlador JSF realiza las siguientes etapas:

● Restaurar los componentes de la vista (restore view). En esta etapa el controlador construye en memoria la estructura de componentes de la página.

● Aplicar los valores de la petición. (apply request values). En esta etapa se recuperan los valores de la request y se asignan a los beans de la página.

● Procesamiento de las validaciones (process validations). Se verifican los parámetros de entrada según un conjunto de reglas definidas en un archivo de configuración.

● Actualizar los valores del modelo (update model values). Los valores leídos y validados son cargados en los beans.

● Invocación a la aplicación (invoke application). Se ejecutan las acciones y eventos solicitados para la página. Si es necesario se realiza la navegación.

● Generación de la página (render response). En esta fase se genera la página que será enviada al usuario con todos sus elementos y valores actualizados.

Gestión de los beans

JSF gestiona automáticamente la creación y el acceso a los beans que utilizan las páginas jsp. Para ello el controlador determina qué beans utiliza la página y dónde debe almacenarlos. El archivo de configuración JSF mapea los nombres cortos de los beans utilizados en las páginas con las clases java que los definen.

¿Cuando se crean los beans?. JSF busca el bean cada vez que se menciona en la página, en el orden en que aparecen en la página. Si el bean no existe en el ámbito, lo crea. Por tanto el orden de las expresiones EL determinan el orden de la creación de los beans, si usamos más de un bean en la página.
¿Cómo se hace esto internamente?. Al procesar la página JSP, las etiquetas JSF añaden código que busca el bean mencionado en cada expresión EL. Si el bean no existe en el ámbito elegido (request, session o application) se crea uno nuevo, llamando a su constructor por defecto, y se asocia al ámbito requerido.
Este mecanismo es fundamental para la comprensión del procesamiento de la página, sobre todo si trabajamos con beans de ámbito request.

El lenguaje de expresiones EL

Para facilitar la visualización y recogida de los datos de los formularios JSF introduce un lenguaje de expresiones similar al introducido en jsp. De hecho a partir de JSF 1.2, ámbos lenguajes de expresiones se han unificado. El lenguaje de expresiones permite acceder de manera muy sencilla a las propiedades de los backbeans.
En una forma más sencilla una expresión EL se puede escribir como:
#{backbean.propiedad} Esto permite asignar o leer valores de las etiquetas JSF.

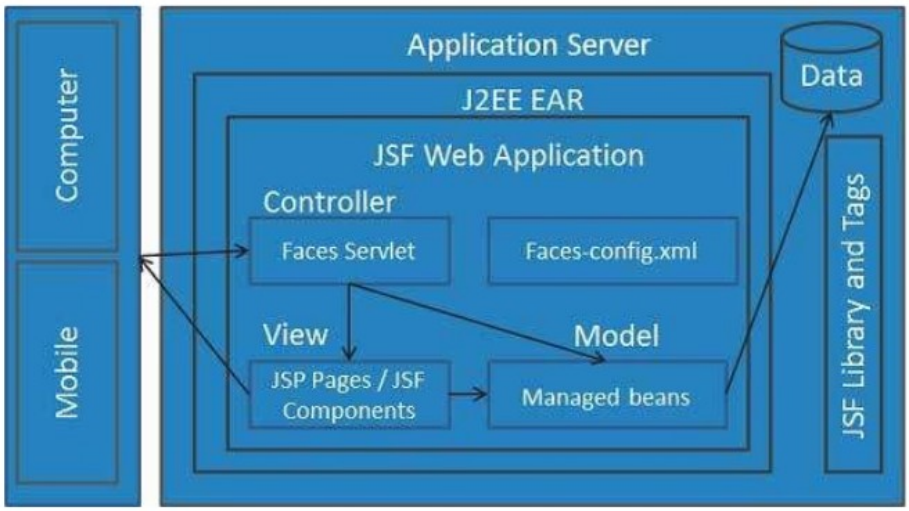
Por ejemplo para escribir y leer valores se pueden usar las etiquetas JSF:

- **<h:outputText value="#{backbean.propiedad}" />**
- **<h:inputTtxt value="#{backbean.propiedad}" />**

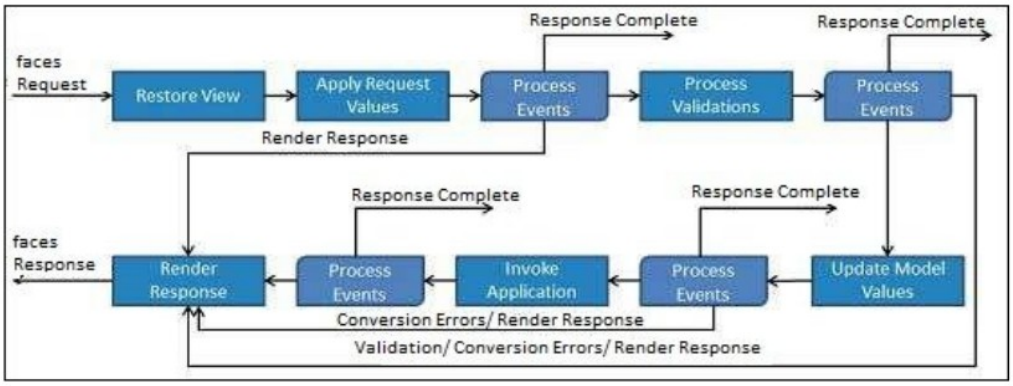
El entorno FacesContext

Un backbean es una clase simple que no conoce nada del resto de la aplicación. Para acceder al entorno JSF y en general al entorno de ejecución en el que la clase se está ejecutando JSF prevé el mecanismo de contexto JSF FacesContext. El FacesContext es una clase que sirve al bean de puente al exterior, ya que le permite acceder no solo al contexto JSF sino también al contexto HTTP. Esto permite al bean el acceso a los demás beans de laaplicación, a las propiedades de la aplicación e incluso a la petición HTTP que se está ejecutando.
El uso del contexto FacesContext desde el bean es simple, ya que la clase FacesContext proporciona un método que devuelve una referencia a la instancia JSF asociada a la petición.

Arquitectura de un aplicación JSF



Flujo de una aplicación JSF



@ManagedBean Annotation

@ManagedBean marks a bean to be a managed bean with the name specified in name attribute. If the name attribute is not specified, then the managed bean name will default to class name portion of the fully qualified class name. In our case, it would be helloWorld.

Another important attribute is **eager**. If eager = "true" then managed bean is created before it is requested for the first time otherwise "lazy" initialization is used in which bean will be created only when it is requested.

Scope Annotations

Scope annotations set the scope into which the managed bean will be placed. If the scope is not specified, then bean will default to request scope. Each scope is briefly discussed in the following table.

S.No	Scope & Description
1	**@RequestScoped**

Bean lives as long as the HTTP request-response lives. It gets created upon a HTTP request and gets destroyed when the HTTP response associated with the HTTP request is finished.

@NoneScoped

2 Bean lives as long as a single EL evaluation. It gets created upon an EL evaluation and gets destroyed immediately after the EL evaluation.

@ViewScoped

3 Bean lives as long as the user is interacting with the same JSF view in the browser window/tab. It gets created upon a HTTP request and gets destroyed once the user postbacks to a different view.

@SessionScoped

4 Bean lives as long as the HTTP session lives. It gets created upon the first HTTP request involving this bean in the session and gets destroyed when the HTTP session is invalidated.

@ApplicationScoped

5 Bean lives as long as the web application lives. It gets created upon the first HTTP request involving this bean in the application (or when the web application starts up and the eager=true attribute is set in @ManagedBean) and gets destroyed when the web application shuts down.

@CustomScoped

6 Bean lives as long as the bean's entry in the custom Map, which is created for this scope lives.