

Tecnologías en el Servidor

Java - JSP

Curso 2020/21
Javier Albert Segui

Índice

- Introducción a JSP
- Ciclo de vida de las páginas JSP
- Elementos JSP
- Manejo de sesiones desde JSP

JSP

- JSP nos permite mezclar, en una misma página, código HTML para la generación de la parte estática de la misma, con contenido dinámico generado a partir de unas marcas especiales. **<% %>**
- El contenido dinámico se obtiene gracias a la posibilidad que tenemos de incrustar dentro de la página de código en JAVA
- Nos permite acceso a Bases de datos remotas para las operaciones CRUD
- Permite la gestión de sesiones en el navegador.

JSP

- El primer borrador de la especificación, por parte de *Sun Microsystems*, vio la luz en el año 1998, apareciendo la especificación 1.0 en el año 1999
- La versión 1.1 apareció a finales del mismo año
- En la actualidad estamos con la versión 2.3.x en producción y con la próxima publicación de la versión 3.0 ya como parte del proyecto Jakarta.
- Toda la información podemos encontrarla en la web del proyecto de la fundación Eclipse:

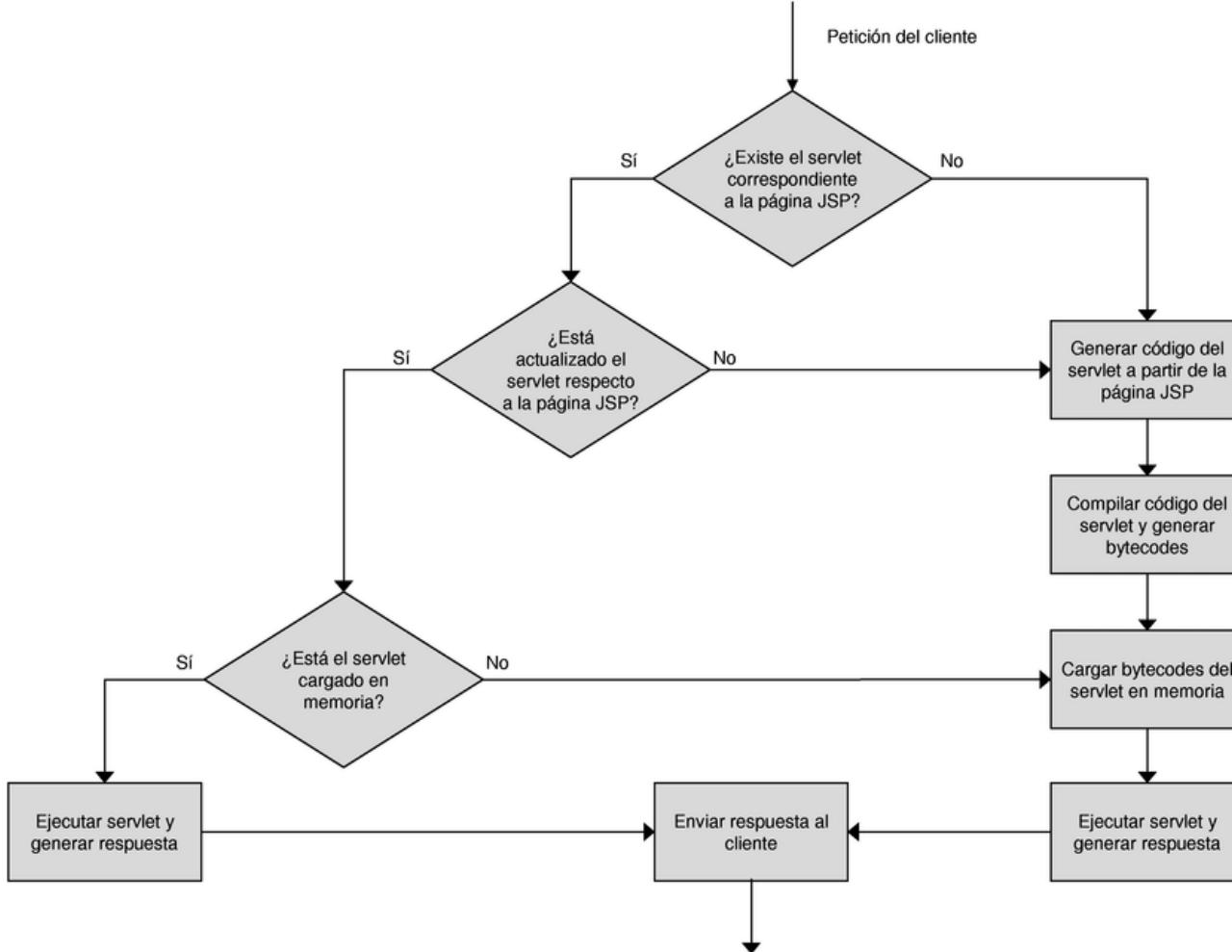
<https://projects.eclipse.org/projects/ee4j.jsp>

JSP

- La página JSP se transforma automáticamente en un servlet.
- Esta conversión se realiza en el servidor donde esta desplegada la página, SOLO se realiza la primera vez que se hace la petición de la pagina, quedando ya compilado para futuros usos.
- El servlet generado será el responsable de procesar cualquier petición a la pagina JSP.
- En caso de realizar cambios en la pagina, se regenera y compila de nuevo el servlet.



JSP



JSP

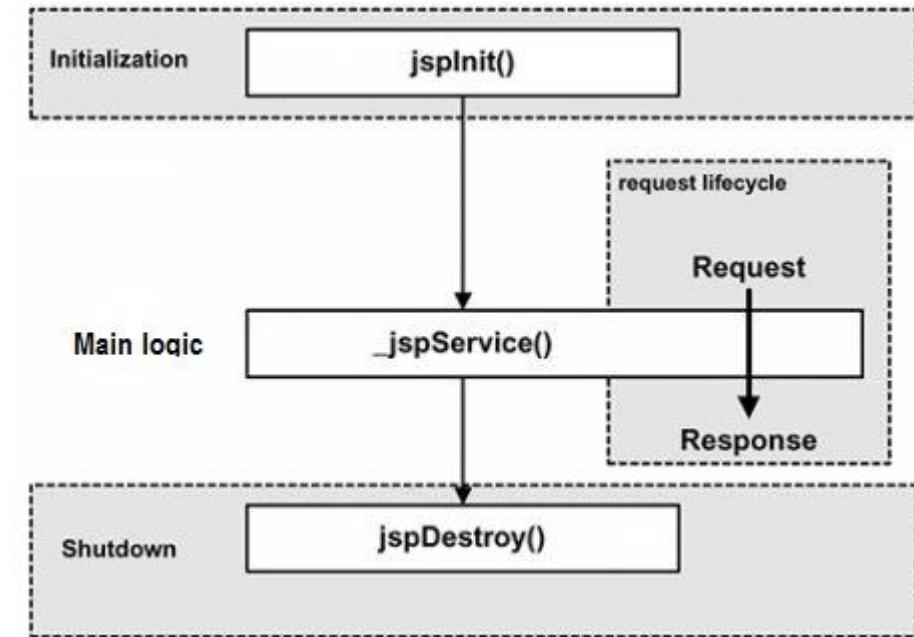
```
<%@ page info="Un ejemplo Hola Mundo" import="java.util.Date"
%>
<HTML>
<head> <title> Hola, Mundo </title> </head>
<body> <h1> ¡Hola, Mundo! </h1>
La fecha de hoy es: <%= new Date().toString() %>
</body>
</HTML>
```

JSP – JSP to Servlet

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse response)
                        throws java.io.IOException, ServletException {
    JspWriter out = null;
    response.setContentType("text/html;ISO-8859-1");
    out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
                Transitional//EN\"");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Hola, Mundo</title>");
    out.println("</head>");
    out.println("<body>");
    out.print("La fecha de hoy es: ");
    out.println(new java.util.Date());
    out.println("</body>");
    out.println("</html>");
}
```

Ciclo de vida JSP

- Cuando llamamos por primera vez y se produce el análisis y compilación de nuestra pagina JSP, se genera un servlet, interno al contenedor de servlets, en el que se implementan las siguientes operaciones:
 - `jsplninit()`
 - Inicializa el servlet generado
 - SOLO se llama en la primera ejecución o cuando se cambia el código de la pagina
 - `_jspService()`
 - Maneja las peticiones.
 - Se invoca en cada petición.
 - `jspDestroy()`
 - Se invoca por el motor para destruir el servlet



Estructura de un JSP

- Desde la especificación JSP 2.0, las páginas JSP pueden ser plantillas de texto sin formato específico o documentos XML, aunque esta nueva notación está aún poco extendida y es algo farragosa.
- La extensión habitual de una página JSP es `.jsp`, aunque se usa `.jspx` para las páginas que son documentos XML y `.jspf` para los fragmentos JSP.
- En la notación habitual (sin usar ningún tipo de scriptlets), una página JSP está compuesta por:
 - Contenido estático: HTML, XML, WML, texto libre, etc.
 - Comentarios JSP: `<%--... --%>`
 - Directivas JSP: `<%@ ...%>`
 - Código EL (Expression Language): `${ ... }`
 - Elementos JSP estándar: `<jsp:...>...</jsp:...>`
 - Elementos custom tags: JSTL, Struts, etc.

Estructura de un JSP

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ page language="java" %>
<%@ page contentType="text/html" %>
<%@ page pageEncoding="ISO-8859-1" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=ISO-8859-1" />
<link rel="stylesheet" type="text/css" href="styles/estilo.css" />
<title>Página JSP de ejemplo</title>
</head>
<%-- sigue en la siguiente transparencia --%>
```

Estructura de un JSP

```
<%-- viene de la transparencia anterior --%>
<body>
    <h1>Ejemplo de página JSP 2.1</h1>
    <jsp:useBean id="errores" scope="session" class="es.us.lsi.daaw.beans.ListaMensajes"
/>
    <div id="div_errores" class="errores">
        <ul>
            <c:forEach var="mensaje" items="${errores.mensajes}">
                <li>${mensaje}</li>
            </c:forEach>
        </ul>
    </div>
</body>
</html>
```

Directivas JSP <%@ page...%>

Atributo	Significado	Ejemplo
import	el equivalente a una sentencia import de Java	<%@ page import="java.util.Date" %>
contentType	genera una cabecera HTTP Content-Type	<%@ page contentType="text/plain" %>
isThreadSafe	si es false, el servlet generado implementará el interface SingleThreadModel (único hilo para todas las peticiones). Por defecto, el valor es true.	
session	Si es false, no se creará un objeto session de manera automática. Por defecto, es true.	
buffer	Define el tamaño del buffer para la salida (en kb), o none si no se desea buffer. Su existencia permite generar cabeceras HTTP o códigos de estado aunque ya se haya comenzado a escribir la salida.	<%@ page buffer="64kb" %>
autoflush	Si es true (valor por defecto), el buffer se envía automáticamente a la salida al llenarse. Si es false, al llenarse el buffer se genera una excepción.	
extends	Permite especificar de qué clase debe descender el servlet generado a partir de la página JSP. No es habitual cambiarlo.	
info	define una cadena que puede obtenerse a través del método getServletInfo	<%@ page info="carro de la compra" %>
errorPage	especifica la página JSP que debe procesar los errores generados y no capturados en la actual.	<%@ page errorPage="error.jsp" %>
isErrorPage	Si es true, indica que la página actúa como página de error para otro JSP. El valor por defecto es false.	
language	Permite especificar el lenguaje de programación usado en el JSP. En la práctica, el lenguaje siempre es Java, por lo que esta directiva no se usa.	
pageEncoding	define el juego de caracteres que usa la página. El valor por defecto es ISO-8859-1.	

Directivas JSP <%@ include...%>

- Incluye un archivo (normalmente un fragmento JSP) antes de que el contenedor transforme la página en un servlet.* Su único atributo es:
 - file = "URL relativa"
 - URL relativa del archivo a incluir. Si empieza por /, se considera relativo al contexto de la aplicación web.

Directivas JSP <%@ taglib...%>

- Permite usar una biblioteca de custom tags. Sus atributos son los siguientes:
 - uri = "URI de la biblioteca"
 - URI que identifica de manera única a una biblioteca de custom tags, p.e. <http://java.sun.com/jsp/jstl/core>.
- prefix = "prefijoBiblioteca"
 - Prefijo para las etiquetas de los elementos definidos en la biblioteca de custom tags. No se admiten prefijos vacíos ni ninguno de los siguientes: jsp, jspx, java, javax, servlet, sun, y sunw.

Directivas JSP <%@ taglib...%>

- La Java Server Pages Standard Tag Library (JSTL), es un conjunto de 5 bibliotecas de *custom tags* estandarizadas:
 - Core (prefijo c): gestión de variables, control de flujo, gestión de URLs.
 - XML (prefijo x): similar a core, pero orientado a XML incluyendo transformación con XSL.
 - i18n (prefijo fmt): internacionalización, incluyendo formateo de fechas y números.
 - SQL (prefijo sql): acceso a bases de datos con SQL, sólo para prototipos (la capa de presentación no debe acceder directamente a los datos).
 - Funciones (prefijo fn): funciones de longitud de colecciones y manipulación de cadenas.

AREA	URI	PREFIJO
Core	http://java.sun.com/jstl/ea/core	c
XML	http://java.sun.com/jstl/ea/xml	x
Internacionalización (I18N)	http://java.sun.com/jstl/ea/fmt	fmt
SQL	http://java.sun.com/jstl/ea/sql	sql

Directivas JSP Lenguaje de Expresiones (EL)

- El Expression Language permite especificar expresiones de forma sencilla en las páginas JSP.
- Las expresiones EL son analizadas y procesadas por el contenedor de servlets al transformar la página JSP en un servlet.
- Una expresión EL tiene la forma \${expresión}, y puede incluir números, cadenas y propiedades de JavaBeans, incluyendo entre otros, los siguientes objetos implícitos* que son tratados como mapas:
 - pageScope: contexto de la página que permite acceder a su mapa y a las propiedades de los objetos servletContext, session, request y response.
 - requestScope: mapa con los atributos de request.
 - sessionScope: mapa con los atributos de sesión.
 - applicationScope: mapa con los atributos de aplicación.
 - param y paramValues: valores enviados por un formulario.

Directivas JSP Lenguaje de Expresiones (EL)

Ejemplos de uso de expresiones EL

- \${param.nombre}
 - El valor del parámetro "nombre" o null si no se ha enviado o es una cadena vacía. Equivale a \${param["nombre"]} y a \${param['nombre']}.
- \${!empty param.direccion}
 - Ciento (true) si el parámetro "dirección" se ha enviado y no es una cadena vacía. Equivalente a \${!empty param["direccion"]}.
- \${sessionScope.carrito.size}
 - El valor de la propiedad "size" de la variable de sesión "carrito". Equivale a \${sessionScope["carrito"].size}.
- \${applicationScope["numUsuarios"]}
 - El valor de la variable de aplicación "numUsuarios". Equivale a \${applicationScope.numUsuarios}.
- - \${carrito.precioTotal}
 - El valor de la propiedad "precioTotal" de la variable "carrito" si la encuentra en los ámbitos de página, request, sesión o aplicación (en ese orden). Si no la encuentra, devuelve null. Equivale a \${carrito["precioTotal"]}.

JSP – Objetos implícitos

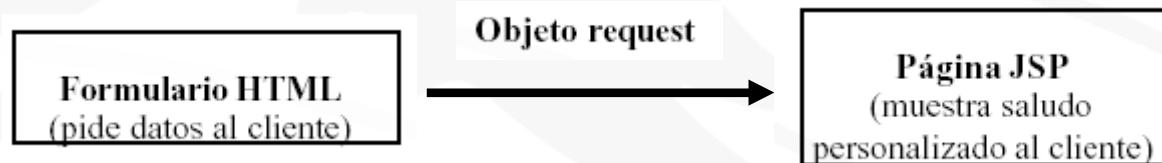
- Los objetos implícitos, son aquellos que están accesibles al motor JSP, por lo que el desarrollador de páginas JSP puede utilizarlos cuando los necesite sin más que invocarlos adecuadamente.
- Son objetos creados por el motor que no necesitan ser declarados para ser usados, sino que se pueden invocar directamente.
- Modelan mucha de la funcionalidad básica de cualquier aplicación web: tratamiento de sesiones, procesamiento de un formulario, etc.

JSP – Objetos implícitos

Objeto	Significado
request	el objeto HttpServletRequest asociado con la petición
response	el objeto HttpServletResponse asociado con la respuesta
out	el Writer empleado para enviar la salida al cliente. La salida de los JSP emplea un buffer que permite que se envíen cabeceras HTTP o códigos de estado, aunque ya se haya empezado a escribir en la salida (out no es un PrintWriter sino un objeto de la clase especial JspWriter).
session	el objeto HttpSession asociado con la petición actual. En JSP, las sesiones se crean automáticamente, de modo que este objeto está instanciado, aunque no se cree explícitamente una sesión.
application	el objeto ServletContext, común a todos los servlets de la aplicación web.
config	el objeto ServletConfig, empleado para leer parámetros de inicialización.
pageContext	permite acceder desde un único objeto a todos los demás objetos implícitos
page	referencia al propio servlet generado (tiene el mismo valor que this). Como tal, en Java no tiene demasiado sentido utilizarla, pero está pensada para el caso en que se utilizará un lenguaje de programación distinto.
exception	Representa un error producido en la aplicación. Solo es accesible si la página se ha designado como página de error (mediante la directiva page isErrorPage).

JSP – Objetos implicitos

Aplicación que pide el nombre al usuario y le devuelve un saludo. Se utiliza un fichero HTML con un formulario que pide los datos al cliente y se los pasa a una página JSP que muestra el saludo con estos datos. El paso de los datos del formulario al JSP se realiza a través de un objeto especial: **request**.



```
<HTML>
<head>
<title> Formulario de petición de nombre
</title>
</head>
<body>
<h1> Formulario de petición de nombre </h1>
<!-- Se envía el formulario al JSP "saludo.jsp" -->
<form method="post" action="saludo.jsp">
<p> Por favor, introduce tu nombre:
<input type="text" name="nombre">
</p>
<p> <input type="submit" value="enviar información"> </form> </body>
</HTML>
```

```
<HTML>
<head>
<title> Formulario de petición de nombre</title>
</head>
<body>
<h1> Formulario de petición de nombre </h1>
<!-- Se envía el formulario al JSP "saludo.jsp" -->
<form method="post" action="saludo.jsp">
<p> Por favor, introduce tu nombre:
<input type="text" name="nombre">
</p>
<p> <input type="submit" value="enviar información">
</form> </body>
</HTML>
```

JSP – Elementos Estándar

- Son 14 elementos que comienzan con el prefijo jsp:
- Algunos tienen que ver con la nueva sintaxis XML de JSP: jsp:root, jsp:body, jsp:attribute, jsp:text, ...
- Otros no se suelen usar mucho, como la inclusión dinámica (jsp:include), o el forwarding (jsp:forward, sin haber generado contenido previamente).
- Los más utilizados son:
 - ←jsp:useBean→: para usar o crear JavaBeans en cualquiera de los mapas de los posibles ámbitos (página, request, sesión o aplicación).
 - ←jsp:setProperty→: para dar valor a las propiedades de los JavaBeans declarados con ←jsp:useBean→.
- Otros como jsp:getProperty han quedado obsoletos con la aparición de las expresiones EL.

JSP – JavaBeans

- Los JavaBeans son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.
- Se usan para encapsular varios objetos en un único objeto (Bean en inglés), para hacer uso de un solo objeto en lugar de varios más simples.
- La especificación de JavaBeans de Sun Microsystems los define como "componentes de software reutilizables que se puedan manipular visualmente en una herramienta de construcción".

JSP - JavaBeans

- Un bean no es más que una clase Java escrita siguiendo unas ciertas convenciones. Estas convenciones hacen posible que herramientas automáticas puedan acceder a sus propiedades y manipularlas sin necesidad de modificar el código. Esto puede servir en el caso de un IDE, por ejemplo, para realizar "programación visual". En JSP el uso principal de los beans es manipular componentes Java sin necesidad de incluir código en la página, accediendo a sus propiedades mediante etiquetas.
- El uso de beans en páginas JSP ofrece diversas ventajas con respecto al uso directo de código Java:
 - Se evita el uso de sintaxis Java, en su lugar se emplean etiquetas con sintaxis XML. Esto permite separar más fácilmente el trabajo de programadores y diseñadores web.
 - Se simplifica la creación y uso de objetos compartidos entre varias páginas.
 - Se simplifica la creación de objetos a partir de los parámetros de la petición

JSP - JavaBeans

- En lo que respecta a su uso con JSP, estas convenciones afectan al modo de definir constructores, métodos y variables miembro:
 - Un bean debe tener al menos un constructor sin argumentos. Este constructor será llamado cuando una página JSP cree una instancia del bean.
 - Un bean no debe tener variables miembros de acceso público. El acceso a las variables y su modificación se debe hacer a través de métodos.
 - El nombre de los métodos de acceso y modificación de variables miembro debe seguir una norma:
 - si la variable tiene el nombre nombreVar, entonces el método de acceso debe llamarse getNombreVar (obsérvese el cambio a mayúsculas de la "N", siguiendo las convenciones habituales de Java), y el método de cambio de valor (en caso de que exista) debe llamarse setNombreVar.
 - En el caso especial de variables booleanas, el método de acceso se debe denominar isNombreVar.

JSP - Gestión de sesiones

- ¡RECORDAR!

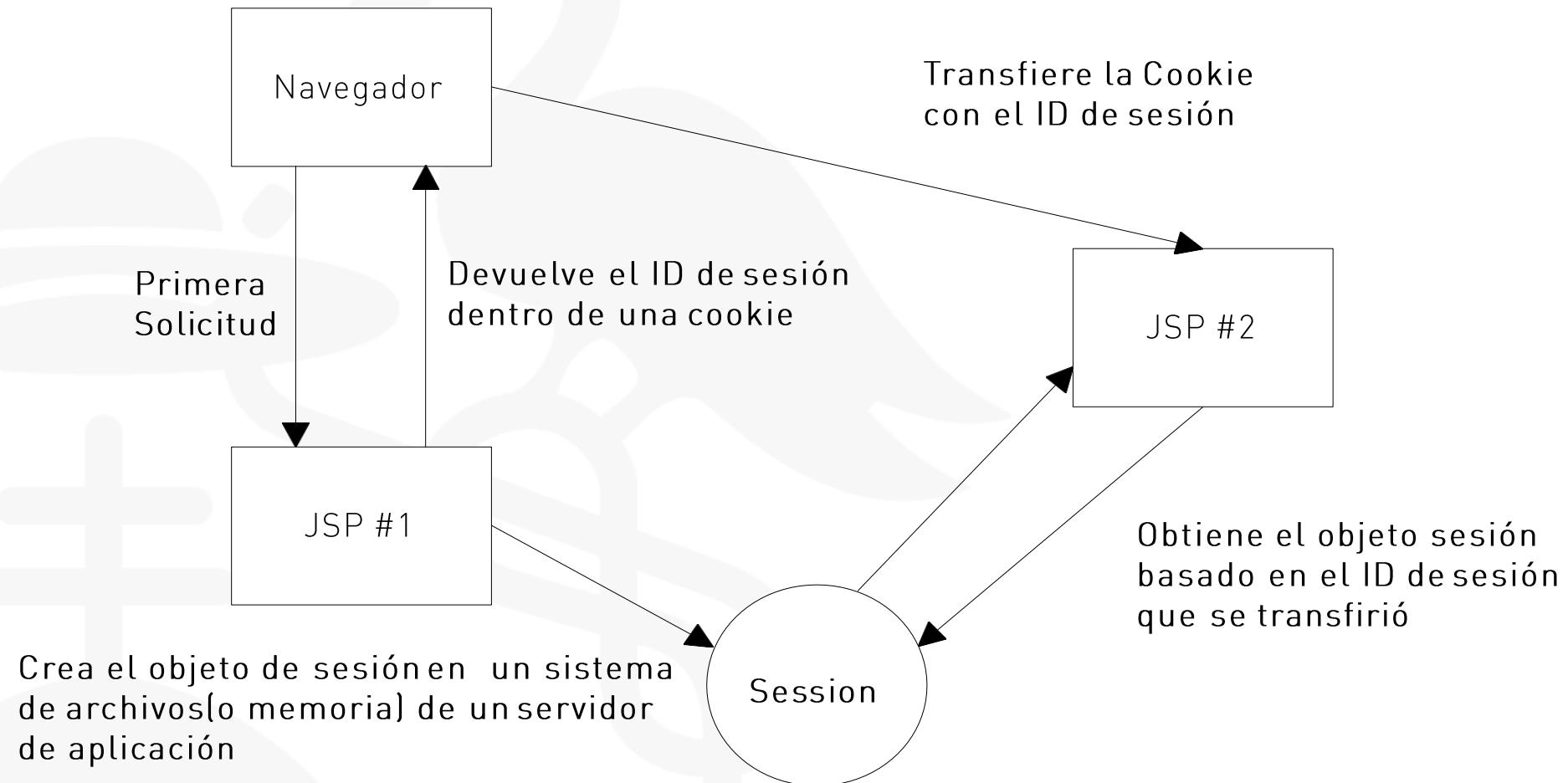
El protocolo HTTP es un protocolo sin estado.

- No posee funcionalidad interna para seguir datos de una solicitud a otra.
- Aunque esto puede proporcionar algunos beneficios en cuanto al rendimiento, también supone un problema en algunas aplicaciones web. Por ejemplo, en aplicaciones tipo: tienda virtual, acceso a un banco, etc.

JSP – Gestión de sesiones

- La tecnología JSP proporciona acceso a un objeto implícito **session**, para almacenar y recuperar valores de una conexión lógica: **session**.
- El objeto **session** es único para cada usuario concreto. La primera vez que un usuario solicita una página jsp, se crea la sesión, esta sesión está identificada por un ID de sesión único y asociado a ella.

JSP – Gestión de sesiones



JSP – Gestión de sesiones

- El objeto session es análogo a la clase HttpSession de la tecnología servlets.
- Una vez obtenida la sesión, normalmente mediante el objeto implícito, es posible:
 - Escribir,
 - Obtener,
 - Eliminar atributos del objeto session.

JSP – Gestión de sesiones

- Escritura:
 - `<% session.setAttribute("nombre", "Usuario pruebas"); %>`
- Lectura:
 - `<%String SNombre=(String) session.getAttribute("nombre"); %>`
 - Observar cómo se asigna el resultado de este método a String. Esto es necesario dado que la sesión, sólo almacena un objeto y devuelve un objeto, sin tener en cuenta el tipo de objeto que esté almacenado. Tenemos que realizar una conversión.
- Eliminar:
 - `<% session.removeAttribute("nombre"); %>`

Tecnologías en el Servidor

Java - MVC

Curso 2020/21

Javier Albert Segui

Índice

- Introducción al patrón MVC
- Implementación del patrón MVC
 - Tecnología Servlets
 - Tecnología JSP
 - Tecnología JDBC

Introducción

- ¡Hacer software no es fácil!
- Diseñar software orientado a objetos es difícil, y diseñar software orientado a objetos reutilizable es todavía más difícil

Chapter 1: Introduction. Design Patterns, The Gang of Four

- Un software capaz de evolucionar tiene que ser reutilizable
- ¡Hay que diseñar para el cambio!
- Para anticiparse a los cambios en los requisitos hay que diseñar pensando en qué aspectos pueden cambiar
- Los patrones de diseño están orientados al cambio

Introducción

- Los patrones de diseño son unas técnicas para resolver problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.
- Un patrón de diseño resulta ser una solución a un problema de diseño.
- Para que una solución sea considerada un patrón debe poseer ciertas características.
 - Debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores.
 - Debe ser reutilizable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Introducción

- Los patrones de diseño pretenden:
 - Proporcionar catálogos de elementos reusables en el diseño de sistemas de software.
 - Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
 - Formalizar un vocabulario común entre diseñadores.
 - Estandarizar el modo en que se realiza el diseño.
 - Facilitar el aprendizaje a las nuevas generaciones de diseñadores condensando conocimiento ya existente.
- Asimismo, no pretenden:
 - Imponer ciertas alternativas de diseño frente a otras.
 - Eliminar la creatividad inherente al proceso de diseño.
- "Abusar o forzar el uso de los patrones puede ser un error"

Patrón MVC

- Es un patrón de arquitectura de las aplicaciones software
- Separa la lógica de negocio de la interfaz de usuario
 - Facilita la evolución por separado de ambos aspectos
 - Incrementa reutilización y flexibilidad
- Utilizado en múltiples frameworks:
 - Java Swing
 - Apache Struts
 - ASP.net MVC
 - Ruby on Rails
 - GTK+
 -

Patrón MVC

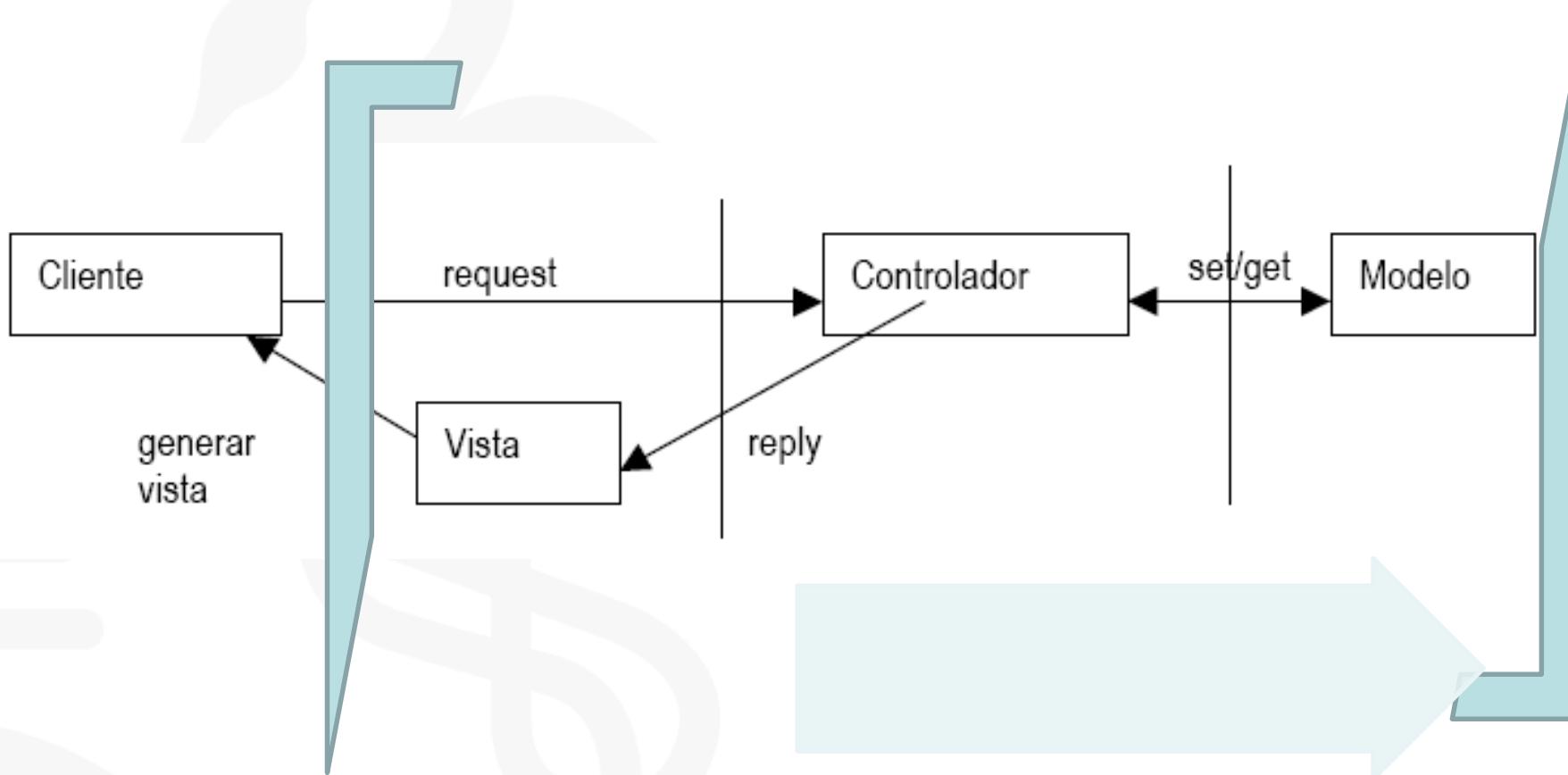
- Modelo – Vista – Controlador
 - Un modelo
 - Varias vistas
 - Varios controladores
- Las vistas y los controladores suelen estar muy relacionados
- Los controladores tratan los eventos que se producen en la interfaz gráfica (vista)
- Esta separación de aspectos de una aplicación da mucha flexibilidad al desarrollador

Patrón MVC

- Flujo de control

- El usuario realiza una acción en la interfaz
- El controlador trata el evento de entrada
 - Previamente se ha registrado
- El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta)
- Se genera una nueva vista. La vista toma los datos del modelo
 - El modelo no tiene conocimiento directo de la vista
- La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo

Patrón MVC



Patrón MVC

- Vista:
 - página HTML
- Controlador:
 - código que obtiene datos dinámicamente y genera el contenido HTML
- Modelo:
 - la información almacenada en una base de datos o en XML junto con las reglas de negocio que transforman esa información (teniendo en cuenta las acciones de los usuarios)

Patrón MVC

- Ventajas:
 - La implementación se realiza de forma modular.
 - Sus vistas muestran información actualizada siempre.
 - Cualquier modificación que afecte al dominio, como aumentar métodos o datos contenidos, implica una modificación sólo en el modelo y las interfaces del mismo con las vistas, no todo el mecanismo de comunicación y de actualización entre modelos.
 - Las modificaciones a las vistas no afectan al modelo de dominio, simplemente se modifica la representación de la información.

Patrón MVC

- Desventajas:

- Para desarrollar una aplicación bajo el patrón de diseño MVC es necesario una mayor dedicación en los tiempos iniciales del desarrollo.
- MVC requiere la existencia de una arquitectura inicial sobre la que se deben construir clases e interfaces para modificar y comunicar los módulos de una aplicación.

Patrón MVC - Modelo

- El **modelo** es un conjunto de clases que representan la información del mundo real que el sistema debe reflejar. Es la parte encargada de representar la lógica de negocio de una aplicación.
- El modelo, a nivel teórico, no debe tener conocimiento acerca de la existencia de las vistas y del controlador.
- **Modelo de Dominio:** Se entiende por modelo de dominio al conjunto de clases definidas a través del análisis de la situación real del problema que queremos solucionar.
- **Modelo de la aplicación:** El modelo de la aplicación es un conjunto de clases que sirven de puente en la relación de las vistas con el modelo de dominio. Tienen conocimiento de las vistas e implementan los mecanismos necesarios para notificar a éstas los cambios

Patrón MVC - Vistas

- Las **vistas** son las encargadas de la representación de los datos, contenidos en el modelo, al usuario.
- La relación entre las vistas y el modelo son de muchas a uno, es decir cada vista se asocia a un modelo, pero pueden existir muchas vistas asociadas al mismo modelo.

Patrón MVC - Controlador

- El controlador es el encargado de interpretar y dar sentido a las instrucciones que realiza el usuario, realizando actuaciones sobre el modelo.
- Si se realiza algún cambio, comienza a actuar, tanto si la modificación se produce en una vista o en el modelo. Interactúa con el Modelo a través de una referencia al propio Modelo.

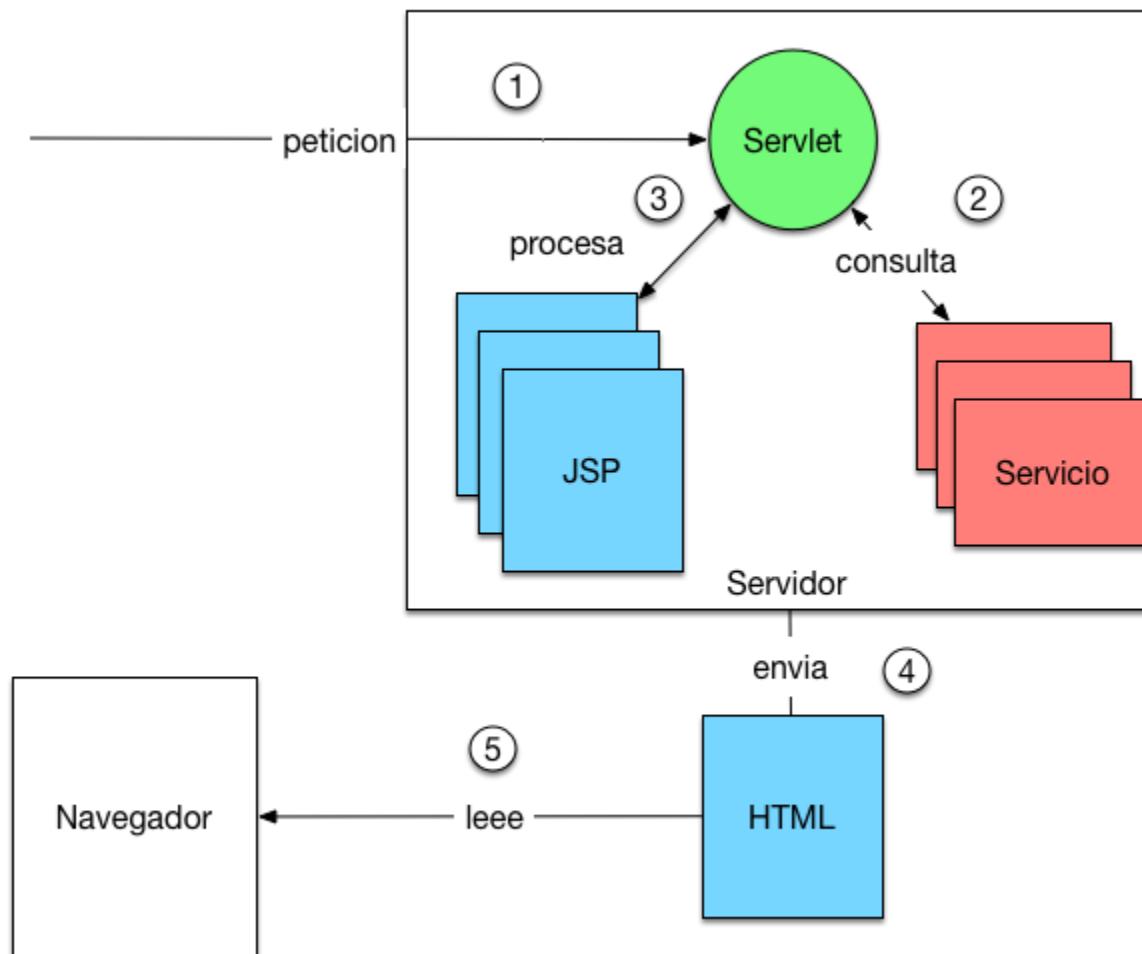
Patrón MVC - Funcionamiento

- Captura de la petición en el controlador
- Procesamiento de la petición
- Generación de respuestas

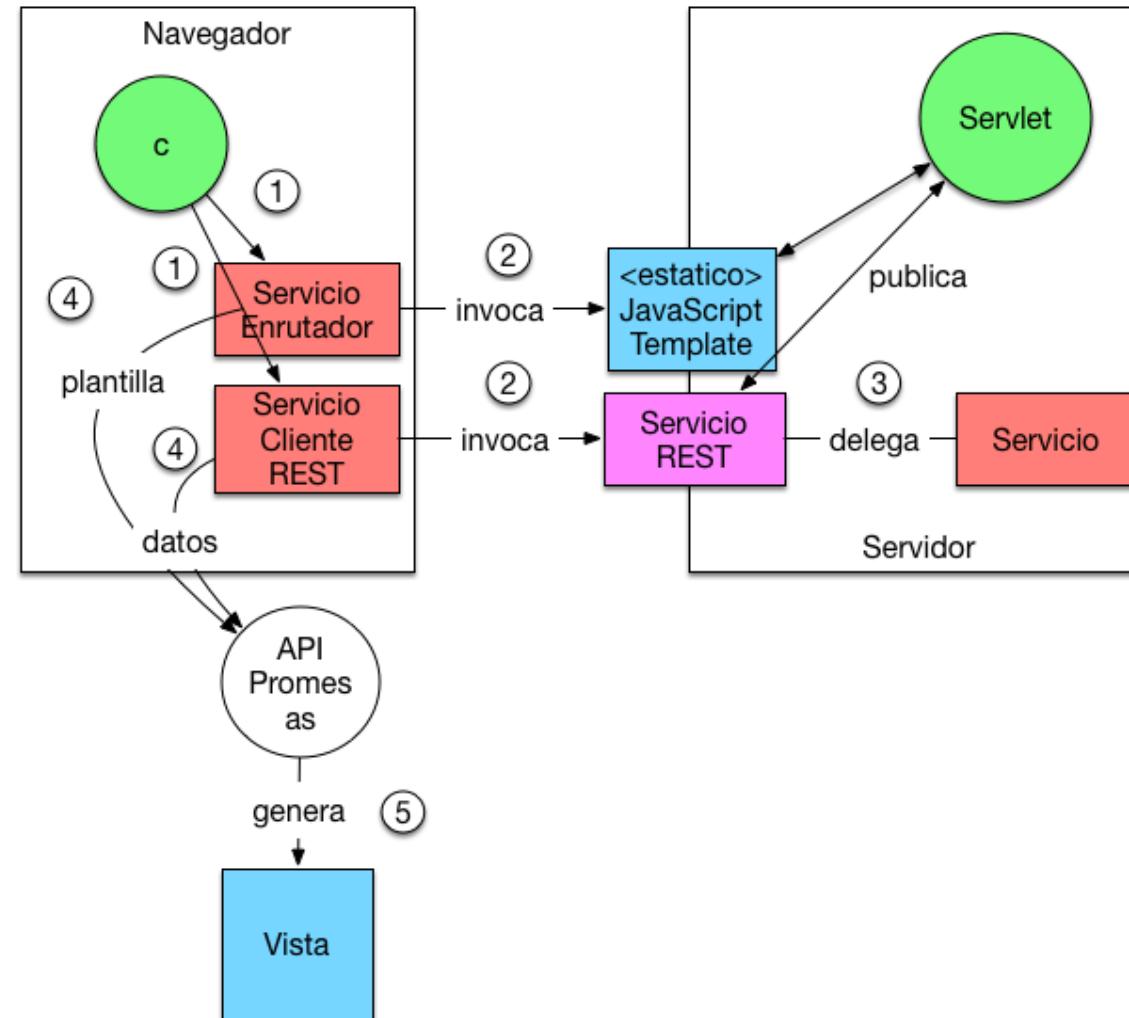
Patrón MVC

- Para implementar el patrón lo primero que tengo que conocer son mis herramientas de trabajo dentro del lenguaje.
- Con un único objetivo: Desarrollar de manera eficiente una aplicación Web.
- No existe una única implementación del patrón de diseño MVC: Struts, JavaServerFaces (JSF), Spring, “la nuestra”, etc.

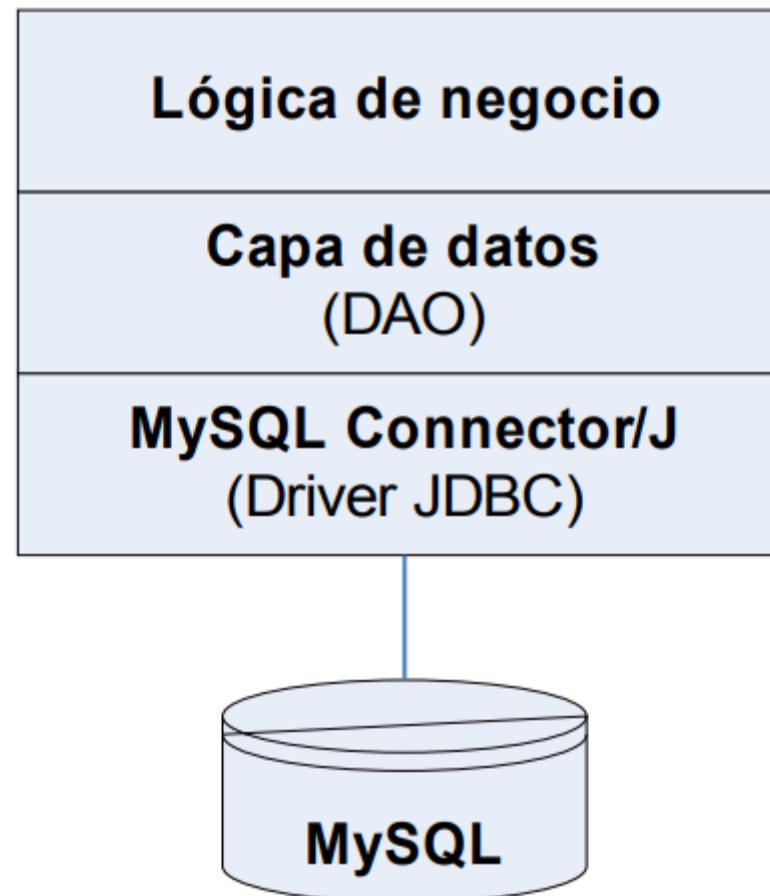
Patrón MVC - Servidor



Patrón MVC - Cliente



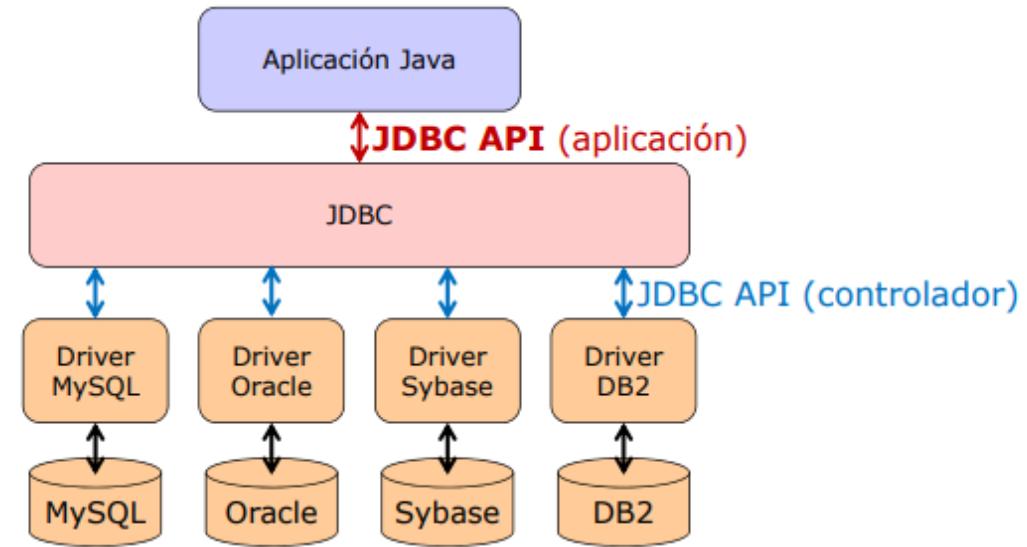
Patrones y Datos



JDBC

es un API (Application programming interface) que describe o define una librería estándar para acceso a fuentes de datos, principalmente orientado a Bases de Datos relacionales que usan SQL (Structured Query Language).

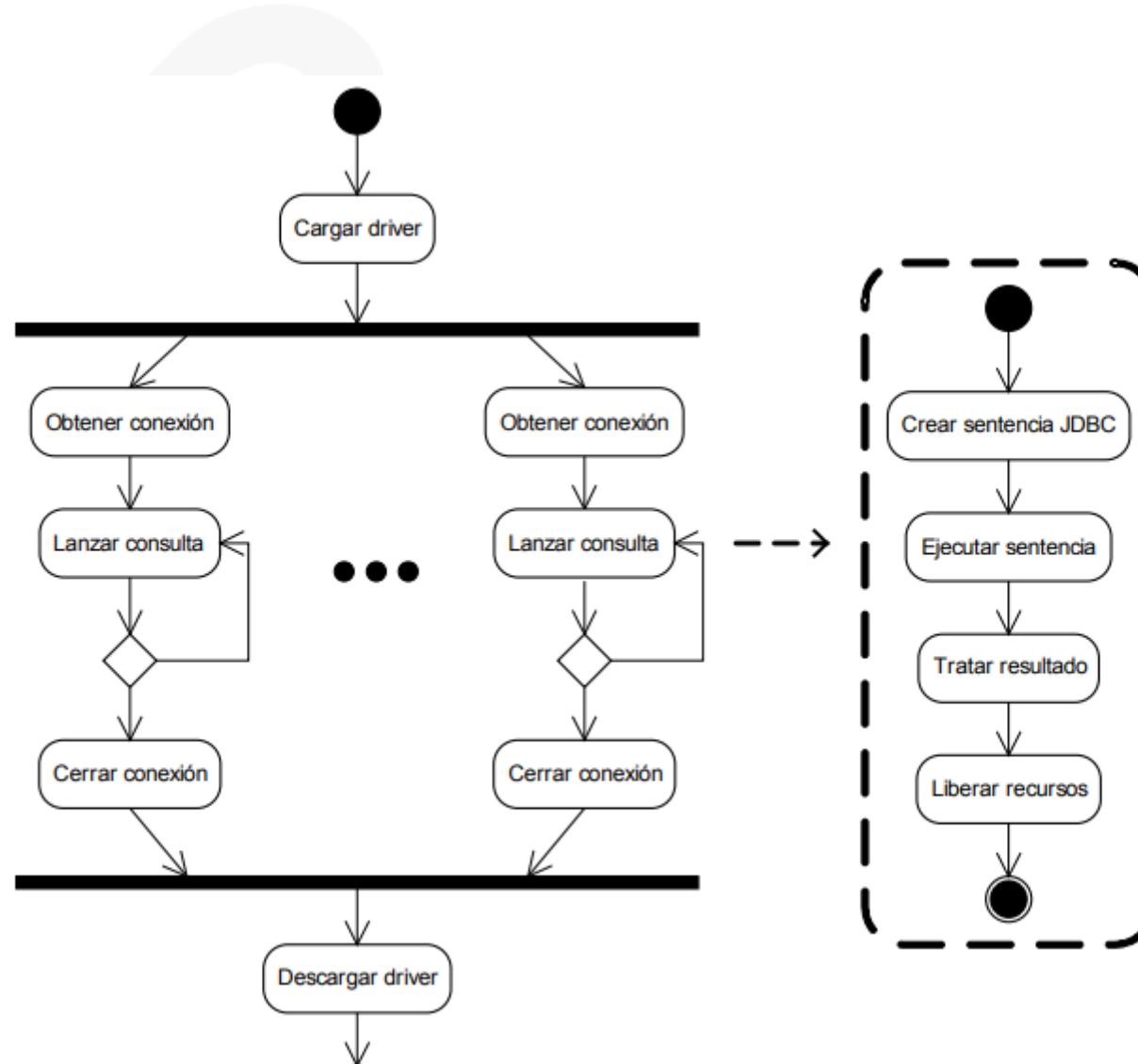
JDBC no sólo provee un interfaz para acceso a motores de bases de datos, sino que también define una arquitectura estándar, para que los fabricantes puedan crear los drivers que permitan a las aplicaciones java el acceso a los datos.



JDBC

- Los drivers JDBC son adaptadores que convierten la petición proveniente del programa JAVA a un protocolo que el SGBD pueda entender.
 - Driver JDBC Tipo 1 (también llamado Puente JDBC-ODBC) convierte el método JDBC a una llamada a una función ODBC. Utiliza los drivers ODBC para conectar con la base de datos.
 - Driver JDBC Tipo 2 (también llamado driver API-Nativo) convierte el método JDBC a llamadas nativas de la API de la base de datos.
 - Driver JDBC Tipo 3. Hace uso de un Middleware entre el JDBC y el SGBD.
 - Driver JDBC Tipo 4 (también llamado Driver Java Puro directo a la base de datos). Es independiente a la plataforma.

JDBC



JDBC

```
Connection conn = DriverManager.getConnection(  
    "jdbc:somejdbcvendor:other data needed by some jdbc vendor",  
    "myLogin",  
    "myPassword");  
try {  
    /* you use the connection here */  
} finally {  
    //It's important to close the connection when you are done with it  
    try {  
        conn.close();  
    } catch (Throwable e) { /* Propagate the original exception  
                           instead of this one that you want just Logged */  
        logger.warn("Could not close JDBC Connection",e);  
    }  
}
```

JDBC

```
try (Statement stmt = conn.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM MyTable")
) {
    while (rs.next()) {
        int numColumns = rs.getMetaData().getColumnCount();
        for (int i = 1; i <= numColumns; i++) {
            // Column numbers start at 1.
            // Also there are many methods on the result set to return
            // the column as a particular type. Refer to the Sun documentation
            // for the list of valid conversions.
            System.out.println( "COLUMN " + i + " = " + rs.getObject(i));
        }
    }
}
```

Tecnologías en el Servidor

Java - Servlets

Curso 2020/21

Javier Albert Segui

Índice

Arquitectura Web - Recordatorio

- La Arquitectura de las aplicaciones Web suelen presentar un esquema de tres niveles:
 - El primer nivel consiste en la capa de presentación que incluye no sólo el navegador, sino también el servidor web que es el responsable de dar a los datos un formato adecuado.
 - El segundo nivel está referido habitualmente a algún tipo de programa o script - lógica de negocio.
 - Finalmente, el tercer nivel proporciona al segundo los datos necesarios para su ejecución.

Java EE

- Java Platform Enterprise Edition (antes J2EE, ahora Jakarta EE) es un estándar para el desarrollo de aplicaciones empresariales (portables, robustas, escalables y seguras) usando tecnología Java.
- Jakarta EE es un conjunto de especificaciones, **no** es un producto. Los productos que cumplen con la especificación son realizados por terceras empresas u organizaciones.
- Toda la información acerca de esta especificación la podemos encontrar en <https://eclipse-ee4j.github.io/jakartaee-platform/>

Jakarta EE

- Ahora la versión de esta especificación es la v8 publicada en septiembre/17
- Incluye:
 - Jakarta WebSocket
 - Jakarta Servlet
 - Jakarta Server Page
 - Jakarta JSON Processing
 - Jakarta Standard Tag Library
 - Jakarta Server Faces
 - Jakarta RESTful Web Services

Tecnologías Java

- En JAVA tenemos dos tecnologías distintas para el Desarrollo de aplicaciones en un entorno Web:
 - Servlets
 - JSP
- Aunque distintas, están íntimamente ligadas, dado que la tecnología JSP al final se transforma en un servlet que son las aplicaciones que se ejecutan sobre el servidor de aplicaciones.

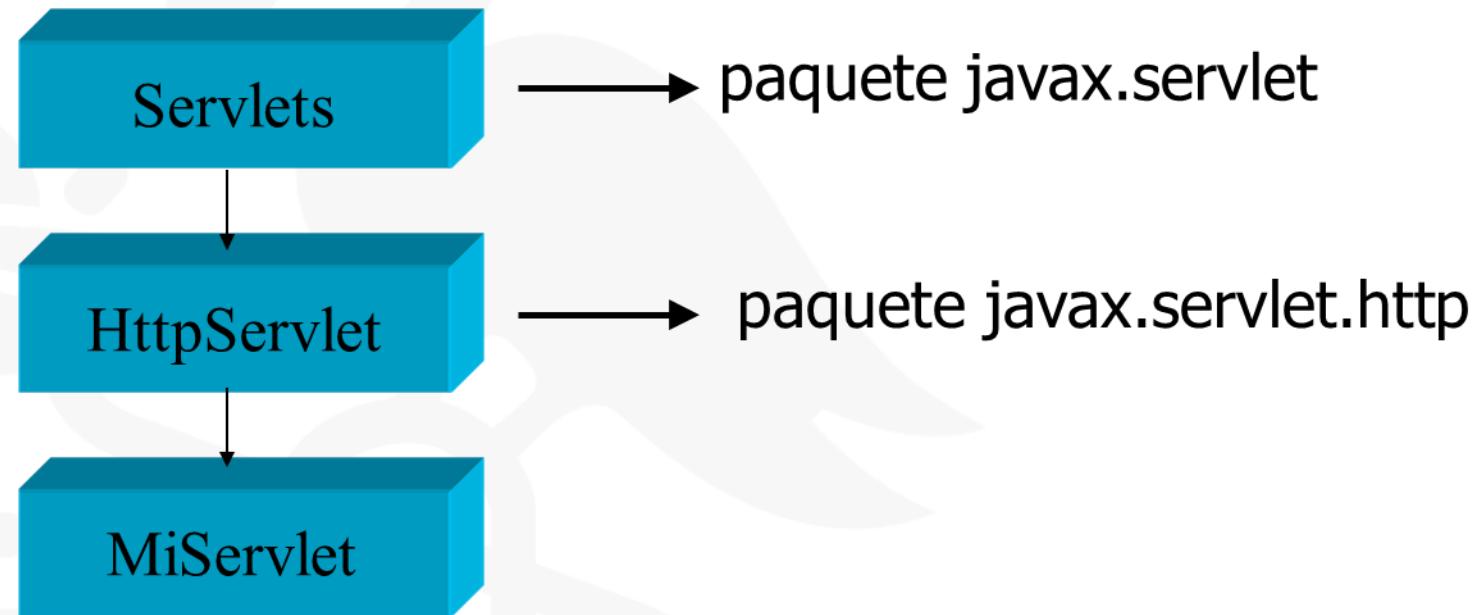
Servlets

- Se puede definir un **Servlet** como un programa JAVA que se ejecuta en un entorno distribuido en red, como un servidor web, y que recibe y responde a las peticiones de un cliente a través del protocolo HTTP.
- Son la contraparte a tecnologías como ASP.net y PHP
- Los **Servlets** son un reemplazo efectivo para los CGI en los servidores que los soporten ya que proporcionan una forma de generar documentos dinámicos utilizando las ventajas de la programación en Java como conexión a base de datos, manejo de peticiones concurrentes, programación distribuida, etc.
 - Por ejemplo, un servlet podría ser responsable de procesar los datos desde un formulario en HTML como registrar la transacción, actualizar una base de datos, contactar algún sistema remoto y retornar un documento dinámico o redirigir a otro servlet u alguna otra cosa.

Servlets

- Los servlets son independientes del servidor y de la Plataforma en que se ejecuta.
- En el servidor tendremos que tener una maquina virtual JAVA, además de las librerías necesarias para el soporte de los servlets.

Servlets - API



Servlets - API

- Hay dos paquetes dentro del API de Servlets
 - javax.servlet
 - javax.servlet.http

- **javax.servlet**

- Interfaz Servlet
 - Interfaz ServletRequest
 - Interfaz ServletResponse
 - Interfaz ServletConfig
 - Interfaz ServletContext
 - Interfaz SingleThreadModel
 - Clase GenericServlet

- javax.servlet.http**

- Interfaz HttpServlet
 - Interfaz HttpServletRequest
 - Interfaz HttpServletResponse
 - Interfaz HttpSession

Servlets - API

○ Clase HttpServlet

- Extiende de la clase GenericServlet y proporciona una implementación de la interfaz Servlet mucho más específica para el protocolo HTTP.
Esta es la clase que extienden la mayoría de los servlets que hay en la actualidad.
- Métodos (throws ServletException, IOException)
 - Gestionan el servicio
 - GET/ POST service (HttpServletRequest req, HttpServletResponse res)
 - GET/ POST processRequest (HttpServletRequest req, HttpServletResponse res) → NetBeans
 - Implementan operaciones propias de HTTP
 - GET doGet (HttpServletRequest req, HttpServletResponse res)
 - POST doPost (HttpServletRequest req, HttpServletResponse res)

Servlet - API

○ `Javax.servlet.http.HttpServletRequest`

- Una interfaz que encapsula la funcionalidad de las peticiones que hace el cliente.
- **IMPORTANTE:** Uno de los métodos de esta interfaz es el método `getParameter (String Name)` utilizado en la mayoría de los servlets que recuperan valores de los de los formularios HTML rellenos por los clientes de las aplicaciones.
- Un objeto `HttpServletRequest` se pasa como parámetro al método `service` de la clase `HttpServlet`.

Servlet - API

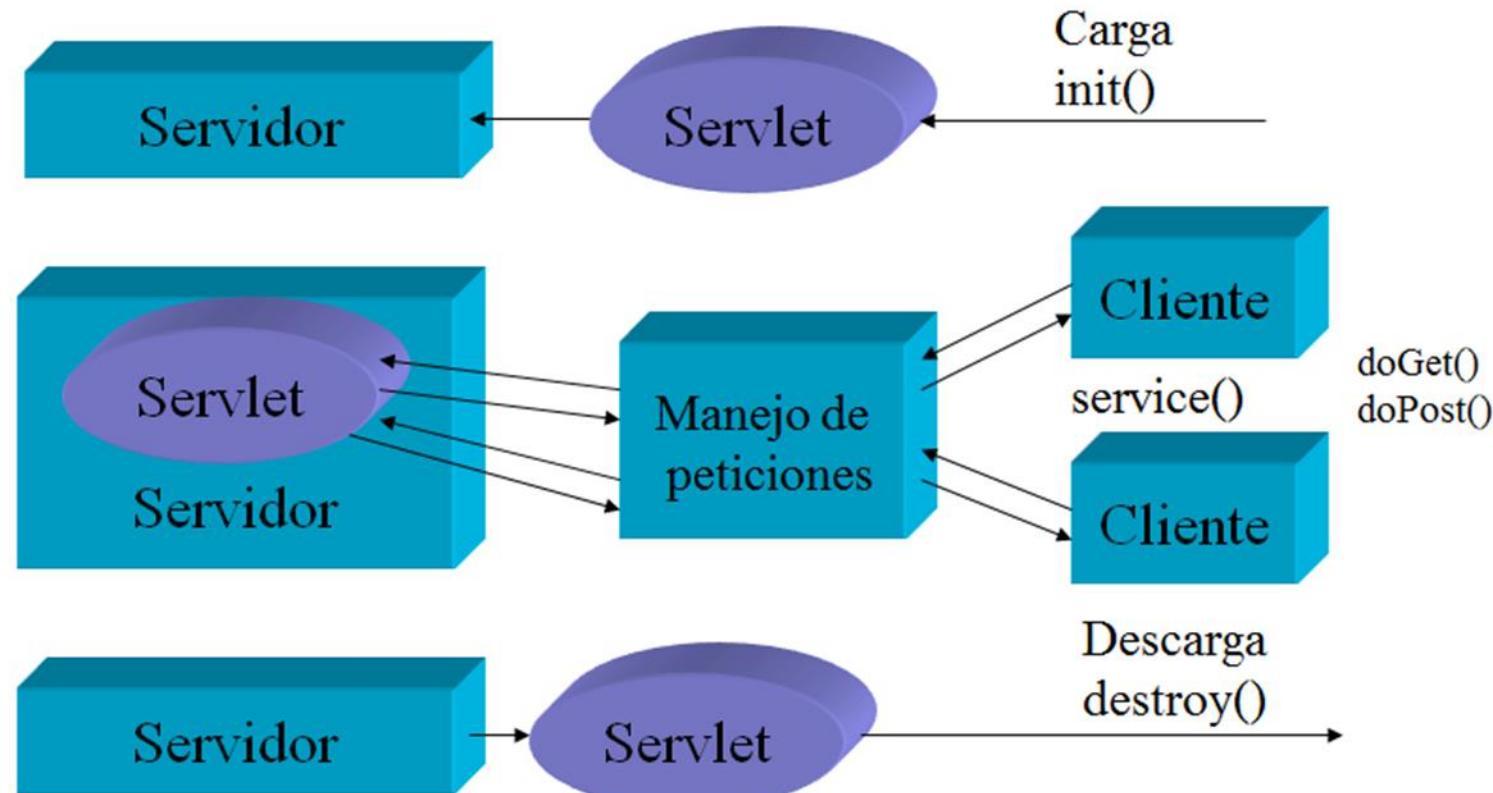
○ `Javax.servlet.http.HttpServletResponse`

- Esta interfaz encapsula la funcionalidad de una respuesta HTTP, incluyendo el manejo de las cabeceras del propio protocolo.
- Interfaz que encapsula la funcionalidad de la respuesta MIME que será enviada al cliente.
- Un objeto HttpServletResponse se pasa como parámetro al método service de la clase HttpServlet.

Servlet – Ciclo de vida

- La interfaz que se utiliza para implementarlos define una serie de métodos para cada una de las etapas.
- El orden es el siguiente:
 - Cuando el servidor carga el Servlet invoca al método **init**.
 - Una vez ejecutado el método init, todas las peticiones serán atendidas por el método **service**.
 - Por último, cuando el servidor web descarga al Servlet se invocará al método **destroy**

Servlet – Ciclo de vida



Acceso a datos

- JDBC, Java DataBase Connectivity, es una API de Java que se usa para acceder a Bases de Datos, tanto locales como remotas.
- Existe una independencia del SGBDR: Si cambia el gestor se minimizan los cambios en la aplicación.
- Lenguaje de consulta SQL.
- El API JDBC (`java.sql`) está formada por cinco grupos:
 - Gestión de Drivers.
 - API para manejadores JDBC.
 - Excepciones.
 - API (trabajar con los datos)
 - Utilidades.

Gestión de Sesiones

- El protocolo HTTP NO posee la capacidad de almacenar estados, (¡es un protocolo sin estado!)
- Posibles soluciones:
 - Cookies.
 - Añadir información en la URL
 - Usar campos ocultos de formularios (HIDDEN)
 - Empleo del objeto HttpSession del servlet.

Gestión de Sesiones

- Los servlets proporcionan una API denominada HttpSession.
- Una interfaz de alto nivel construida sobre cookies y la reescritura de las urls (de manera transparente para el desarrollador).
- Permite almacenar objetos de cualquier tipo.

Gestión de Sesiones

- Pasos para el manejo de sesiones con Servlets
 - Crear un objeto HttpSession: Usaremos el método getSession de HttpServletRequest
 - Añadir / Recuperar información:
 - getAttribute("nombre_variable"): devuelve un Object en caso de que la sesión tenga ya contenido en esa variable, null si no se ha utilizado nunca.
 - setAttribute("nombre_variable",referencia): OJO si el objeto existe, se sobreescribe.
 - getAttributes():retorna un tipo Enumeration con los nombres de todos los atributos establecidos en la sesión del usuario.

Servicios Web y microservicios

Javier Albert Segui
Curso 2020 / 21

Servicios Web

- Del inglés, Web Services.
- Es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.
- La interoperabilidad entre distintos sistemas, lenguajes y plataformas se consigue mediante la adopción de unos estándares abiertos.

Servicios Web

Según el W3C un servicio web es:

“Un sistema software diseñado para soportar la interacción M2M, a través de una red, de forma interoperable.”

Cuenta con una interfaz descrita en un formato procesable por un equipo informático (específicamente en **WSDL**), a través de la que es posible interactuar con el mismo mediante el intercambio de mensajes **SOAP**, típicamente transmitidos usando serialización **XML** sobre **HTTP** conjuntamente con otros estándares web.

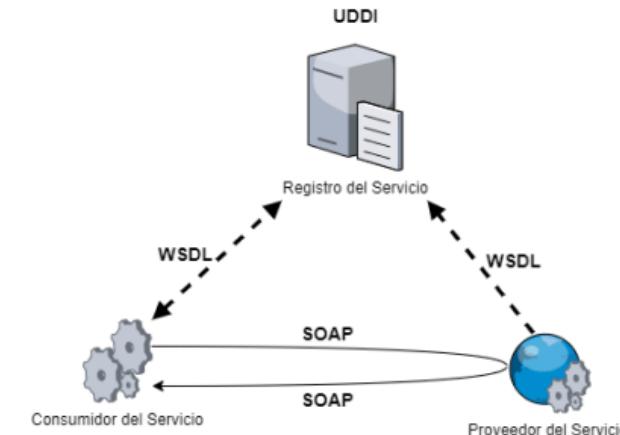
Servicios Web

- Las características deseables de un Servicio Web son:
 - **Un servicio debe poder ser accesible a través de la Web.** Para ello debe utilizar protocolos de transporte estándares como HTTP, y codificar los mensajes en un lenguaje estándar que pueda conocer cualquier cliente que quiera utilizar el servicio.
 - **Un servicio debe contener una descripción de sí mismo.** De esta forma, una aplicación podrá saber cuál es la función de un determinado Servicio Web, y cuál es su interfaz, de manera que pueda ser utilizado de forma automática por cualquier aplicación, sin la intervención del usuario.
 - **Debe poder ser localizado.** Deberemos tener algún mecanismo que nos permita encontrar un Servicio Web que realice una determinada función. De esta forma tendremos la posibilidad de que una aplicación localice el servicio que necesite de forma automática, sin tener que conocerlo previamente el usuario.

Servicios Web - Arquitectura

- Arquitectura funcional: Podemos distinguir 3 agentes distintos en funciones.

Proveedor de servicio	Implementa unas determinadas operaciones (servicio). Un cliente podrá solicitar uno de estos servicios a este proveedor.
Cliente del servicio	Invoca a un proveedor de servicio para la realización de alguna de las operaciones que proporciona.
Registro de servicios	Mantiene una lista de proveedores de servicios disponibles, junto a sus descripciones.



Servicios Web - Arquitectura

○ Arquitectura de capas de protocolos:

Capa	Descripción
Transporte de servicios	Es la capa que se encarga de transportar los mensajes entre aplicaciones. Normalmente se utiliza el protocolo HTTP para este transporte, aunque los servicios web pueden viajar mediante otros protocolos de transferencia de hipertexto como SMTP, FTP o BEEP.
Mensajería XML	Es la capa responsable de codificar los mensajes en XML de forma que puedan ser entendidos por cualquier aplicación. Puede implementar los protocolos XML-RPC o SOAP.
Descripción de servicios	Se encarga de definir la interfaz pública de un determinado servicio. Esta definición se realiza mediante WSDL.
Localización de servicios	Se encarga del registro centralizado de servicios, permitiendo que estos sean anunciados y localizados. Para ello se utiliza el protocolo UDDI.

Servicios Web

- Hay 2 tipos de Servicios Web:
 - SOAP
 - RESTful

Servicios Web – SOAP

○ SOAP – Simple Object Access Protocol

- Es un protocolo escrito en XML para el intercambio de información entre aplicaciones. Es un formato para enviar mensajes, diseñado especialmente para servir de comunicación en Internet, pudiendo extender los HTTP headers. Es una forma de definir qué información se envía y cómo mediante XML.
- Dos tipos de mensajes:
 - Mensajes orientados al documento: Contienen cualquier tipo de contenido que queramos enviar entre aplicaciones.
 - Mensajes orientados a RPC: Este tipo de mensajes servirá para invocar procedimientos de forma remota (Remote Procedure Calls). Podemos verlo como un tipo más concreto dentro del tipo anterior, ya que en este caso como contenido del mensaje especificaremos el método que queremos invocar junto a los parámetros que le pasamos, y el servidor nos deberá devolver como respuesta un mensaje SOAP con el resultado de invocar el método.

Servicios Web – SOAP



```
<SOAP-ENV:Envelope  
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">  
    <SOAP-ENV:Body>  
        <ns:getTemperatura xmlns:ns="http://rvg.uah.es/ns">  
            <area>Alcala de Henares</area>  
        </ns:getTemperatura>  
    </SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

Servicios Web – SOAP

○ WSDL - Web Services Description Language

- Es un lenguaje basado en XML para describir los servicios web y cómo acceder a ellos. Es el formato estándar para describir un web service, y fue diseñado por Microsoft e IBM. WSDL es una parte integral del estándar UDDI, y es el lenguaje que éste utiliza.
- El fichero WSDL describirá la interfaz del Servicio Web, con los métodos a los que podemos invocar, los parámetros que debemos proporcionarles y los tipos de datos de dichos parámetros.

Servicios Web – SOAP

- El elemento raíz dentro de este fichero es **definitions**, donde se especifican los espacios de nombres que utilizamos en nuestro servicio. Dentro de este elemento raíz encontramos los siguientes elementos:
 - **types**: Se utiliza para definir los tipos de datos que se intercambiarán en el mensaje.
 - **message**: Define los distintos mensajes que se intercambiarán durante el proceso de invocación del servicio. Se deberán definir los mensajes de entrada y salida para cada operación que ofrezca el servicio. En el caso de mensajes RPC, en el mensaje de entrada se definirán los tipos de parámetros que se proporcionan, y en el de salida el tipo del valor devuelto.
 - **portType**: Define las operaciones que ofrece el servicio. De cada operación indica cuáles son los mensajes de entrada y salida, de entre los mensajes definidos en el apartado anterior.
 - **binding**: Indica el protocolo y el formato de los datos para cada mensaje de los definidos anteriormente. Este formato puede ser orientado al documento u orientado a RPC. Si es orientado al documento tanto el mensaje de entrada como el de salida contendrán un documento XML. Si es orientado a RPC el mensaje de entrada contendrá el método invocado y sus parámetros, y el de salida el resultado de invocar dicho método.
 - **service**: Define el servicio como una colección de puertos a los que se puede acceder. Un puerto es la dirección (URL) donde el servicio actúa. Esta será la dirección a la que las aplicaciones deberán conectarse para acceder al servicio. Además, contiene la documentación en lenguaje natural del servicio.

Servicios Web – SOAP

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:tns="http://rvg.ua.es/wsdl"
    xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
    targetNamespace="http://rvg.ua.es/wsdl"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
    <message name="getTempRequest">
        <part name="string_1"
            xmlns:partns="http://www.w3.org/2001/XMLSchema"
            type="partns:string" />
    </message>
    <message name="getTempResponse">
        <part name="double_1"
            xmlns:partns="http://www.w3.org/2001/XMLSchema"
            type="partns:double" />
    </message>
    <portType name="TempPortType">
        <operation name="getTemp">
            <input message="tns:getTempRequest" />
            <output message="tns:getTempResponse" />
        </operation>
    </portType>
    <binding name="TempPortSoapBinding" type="tns:TempPortType">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="getTemp">
            <soap:operation soapAction="" style="rpc" />
            <input>
                <soap:body use="encoded"
                    namespace="http://rvg.ua.es/wsdl"
                    encodingStyle=
                        "http://schemas.xmlsoap.org/soap/encoding/" />
            </input>
            <output>
                <soap:body use="encoded"
                    namespace="http://rvg.ua.es/wsdl"
                    encodingStyle=
                        "http://schemas.xmlsoap.org/soap/encoding/" />
            </output>
        </operation>
    </binding>
    <service name="Temp">
        <documentation>Documentacion</documentation>
        <port name="TempPort" binding="tns:TempPortSoapBinding">
            <soap:address
                location="http://localhost:7001/sw_temp/Temp" />
        </port>
    </service>
</definitions>
```

Servicios Web – SOAP

- UDDI - Universal Description, Discovery and Integration
 - Es un estándar XML para describir, publicar y encontrar servicios web. Es un directorio donde las compañías pueden registrar y buscar servicios web. Es un directorio de interfaces de servicios web descritos en WSDL que se comunican mediante SOAP.

Servicios Web – REST

- RESTful: son un tipo de Servicios web que se adhieren a una serie de restricciones arquitectónicas englobadas bajo REST.
- REST: proviene de la tesis doctoral de Roy Fielding, publicada en el año 2000, y significa REpresentational State Transfer
- REST es un conjunto de restricciones que, cuando son aplicadas al diseño de un sistema, crean un estilo arquitectónico de software.

Servicios Web – REST

- Debe ser un sistema cliente-servidor
- Tiene que ser sin estado, es decir, no hay necesidad de que los servicios guarden las sesiones de los usuarios (cada petición al servicio tiene que ser independiente de las demás)
- Debe soportar un sistema de cachés: la infraestructura de la red debería soportar caché en diferentes niveles
- Debe ser un sistema uniformemente accesible (con una interfaz uniforme): Esta restricción define cómo debe ser la interfaz entre clientes y servidores. La idea es simplificar y desacoplar la arquitectura, permitiendo que cada una de sus partes puede evolucionar de forma independiente
- Tiene que ser un sistema por capas: un cliente no puede "discernir" si está accediendo directamente al servidor, o a algún intermediario. Las "capas" intermedias van a permitir soportar la escalabilidad, así como reforzar las políticas de seguridad

Servicios Web – REST

- Un recurso REST es cualquier cosa que sea direccionable (y, por lo tanto, accesible) a través de la Web. Por direccionable nos referimos a recursos que puedan ser accedidos y transferidos entre clientes y servidores.
- La representación de los recursos es lo que se envía entre los servidores y clientes. Una representación muestra el estado temporal del dato real almacenado en algún dispositivo de almacenamiento en el momento de la petición
- Direccionalidad de los recursos: URI, o Uniform Resource Identifier, en un servicio web RESTful es un hiperenlace a un recurso, y es la única forma de intercambiar representaciones entre clientes y servidores. Un servicio web RESTful expone un conjunto de recursos que identifican los objetivos de la interacción con sus clientes.

Servicios Web - REST

- Ventajas:

- Separación entre Cliente y Servidor
- Visibilidad, fiabilidad y escalabilidad
- Independencia del lenguaje y Plataforma
- Experiencia de usuario
- Menor consumo de Recursos en el servidor

- Desventajas:

- Complejidad aumentada
- Centralización de la información

Servicios Web - SOAP vs REST

Ventajas REST	Ventajas SOAP
·Pocas operaciones con muchos recursos	·Muchas operaciones con pocos recursos
·Se centra en la escalabilidad y rendimiento a gran escala para sistemas distribuidos hipermedia	·Se centra en el diseño de aplicaciones distribuidas
·HTTP GET, HTTP POST, HTTP PUT, HTTP DEL	·SMTP,HTTP POST, MQ
·XML auto descriptivo	·Tipado fuerte, XML Schema
·Síncrono	·Síncrono y Asíncrono
·HTTPS	·WS SECURITY
·Comunicación punto a punto y segura	·Comunicación origen a destino seguro

Servicios Web - Seguridad

- En la utilización de los Servicios Web, encontramos problemas de seguridad en diferentes aspectos. Podemos encontrar problemas de seguridad en cuanto a la confidencialidad, la autentificación y la seguridad de la red.
- **Confidencialidad:**
 - Cuando un cliente utiliza un Servicio Web, deberá enviarle un mensaje a este servicio a través de la red, y el servicio le responderá mediante otro mensaje. Estos mensajes contendrán información que puede ser confidencial.
 - Dado que estos mensajes se envían mediante protocolo HTTP, podrán ser encriptados mediante SSL evitando de esta forma que puedan ser interceptados por un tercero.
- **Autentificación:** Puede que necesitemos identificar a un usuario para prestarle un determinado servicio, o bien para saber si tiene autorización para acceder a dicho servicio.
 - Podemos utilizar para ello la autentificación que nos proporciona el protocolo HTTP. Encontramos el mismo problema que en el caso anterior, necesitamos invocar un conjunto de servicios, deberemos autenticarnos por separado para cada uno de ellos, ya que pueden estar distribuidos en distintos servidores a través de Internet. Para solucionar este problema, deberíamos contar con un contexto compartido global de donde cualquier servicio pudiese obtener esta información de autentificación (p.e. → SSO).
- **Seguridad de la red:** Hemos de pensar que estamos permitiendo invocar procedimientos remotos mediante protocolo HTTP, que en un principio fue diseñado para la extracción de documentos. Por lo tanto, sus puertos no suelen ser cortados por ningún firewall, de forma cualquiera podrá utilizar estos servicios libremente, sin que los firewalls puedan controlarlo.

API

- Es una abreviatura de Application Programming Interfaces, que en español significa **interfaz de programación de aplicaciones**.
- Se trata de un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones, permitiendo la comunicación entre dos aplicaciones de software a través de un conjunto de reglas.
- Podemos encontrar estas API en:
 - Navegador: Geolocalización, Storage, WebWorkers....
 - Servidor: nos dan servicios de terceros de una forma sencilla para nuestras aplicaciones.

API

- Actúa como intermediario virtual, remite información de una interfaz, como una aplicación móvil, a otra. Las API conectan diferentes partes de una plataforma de software con el fin de garantizar que la información acabe en el lugar adecuado.
- Estos puntos de conexión no solo funcionan como canales para las comunicaciones internas, sino también como un modo para que las herramientas externas accedan a la misma información. Así pues, las API pueden incluirse en una de las dos categorías:
 - API privado
 - API abierto



API

- API privadas:

- Solo pueden acceder los desarrolladores y los usuarios de la organización. Estas API normalmente conectan procesos internos de los equipos con el fin de reducir el trabajo aislado y mejorar la colaboración.

- API Abiertas

- proporcionan a los desarrolladores externos un modo de acceder fácilmente a la información e integrarla entre herramientas. Una API abierta o pública ahorra a los desarrolladores tiempo, pues les permite conectar su plataforma con herramientas que ya tienen, lo que reduce la necesidad de crear funciones totalmente nuevas.

API

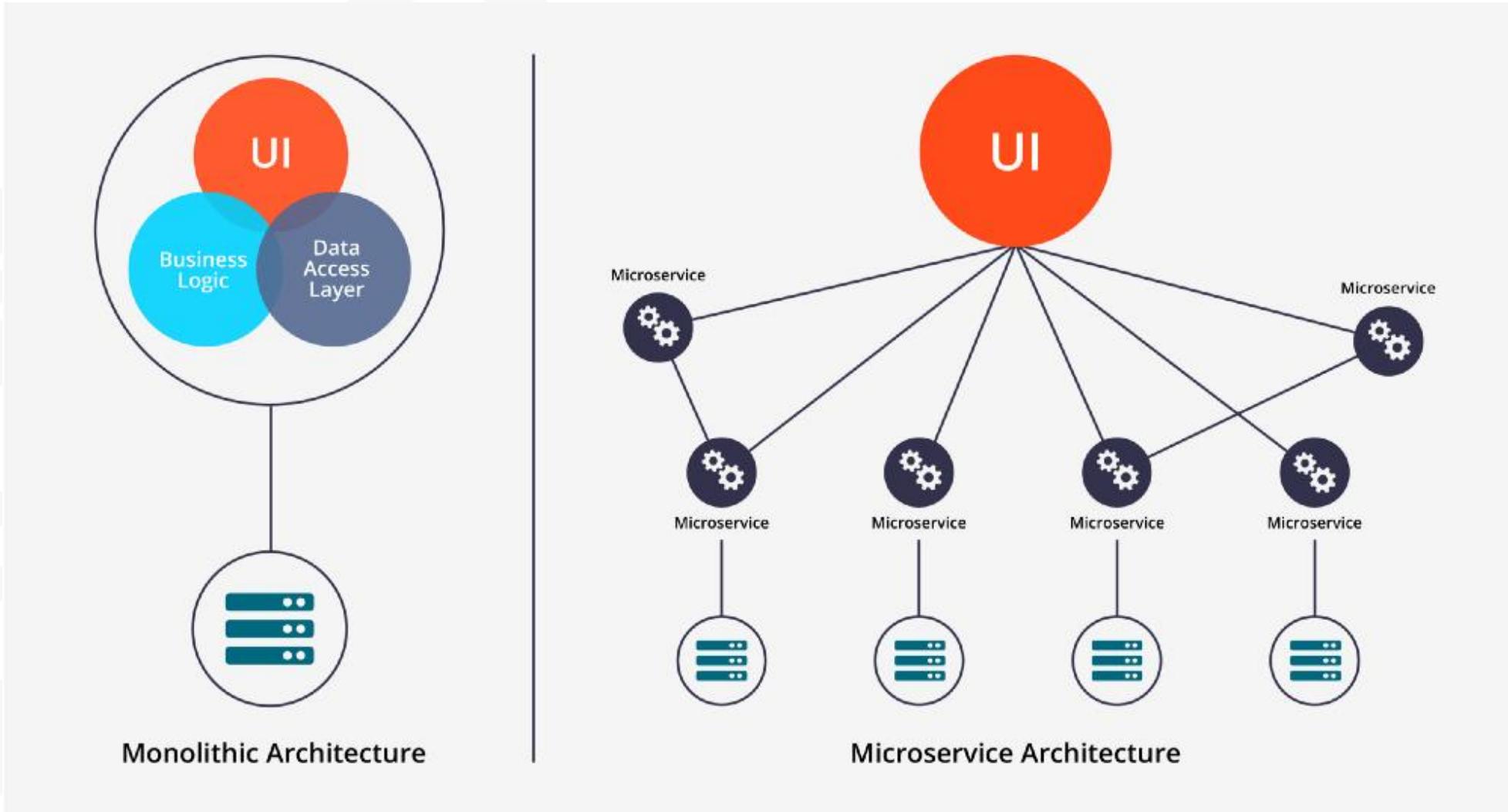
ose puede utilizar para conectar casi todos los procesos. A continuación, encontrarás algunos ejemplos de las funciones habituales de las API:

- Compartir información de vuelos entre líneas aéreas y sitios de viajes
- Usar Google Maps en una aplicación de uso compartido de transporte
- Crear bots de chat en un servicio de mensajería
- Integrar vídeos de YouTube en una página web
- Automatizar flujos de trabajo entre herramientas de software B2B

Microservicios

- es una aproximación para el **desarrollo de software** que consiste en construir una aplicación como un conjunto de pequeños servicios, los cuales se ejecutan en su propio proceso y se comunican con mecanismos ligeros (normalmente una API de recursos HTTP).
- Cada servicio se encarga de implementar una funcionalidad completa del negocio.
- Cada servicio es desplegado de forma independiente y puede estar programado en distintos lenguajes y usar diferentes tecnologías de almacenamiento de datos.

Microservicios



Microservicios - Características

- Los componentes son Servicios.
- Organizada en torno a las funcionalidades del negocio
- Extremos inteligentes
- Gobierno descentralizado
- Gestión de datos descentralizada
- Diseño tolerante a fallos
- Automatización de la infraestructura
- Autonomía
- Especialización

Microservicios - Ventajas

- Versátil — los microservicios permiten el uso de diferentes tecnologías y lenguajes.
- Fácil de integrar y escalar con aplicaciones de terceros.
- Los microservicios pueden desplegarse según sea necesario, por lo que funcionan bien dentro de metodologías ágiles.
- Las soluciones desarrolladas con arquitectura de microservicio permiten la mejora rápida y continua de cada funcionalidad.
- El mantenimiento es más simple y barato — con los microservicios se puede hacer mejoras de un módulo a la vez, dejando que el resto funcione normalmente.
- El desarrollador puede aprovechar las funcionalidades que ya han sido desarrolladas por terceros — no necesita reinventar la rueda, simplemente utilizar lo que ya existe y funciona.
- Un proyecto modular basado en microservicios evoluciona de forma más natural, es una forma fácil de gestionar diferentes desarrollos, utilizando los recursos disponibles, al mismo tiempo

Microservicios - Desventajas

- Debido a que los componentes están distribuidos, las pruebas globales son más complicadas.
- Es necesario controlar el número de microservicios que se gestionan, ya que cuantos más microservicios existan en una solución, más difícil será gestionarlos e integrarlos.
- Los microservicios requieren desarrolladores experimentados con un nivel muy alto de experiencia.
- Se requiere un control exhaustivo de la versión.
- La arquitectura de microservicios puede ser costosa de implementar debido a los costos de licenciamiento de aplicaciones de terceros.

Usabilidad y accesibilidad

Javier Albert Segui
Curso 2020 / 21

Usabilidad

- Del inglés *usability* se refiere a “la facilidad con que las personas pueden utilizar una herramienta en particular con el fin de alcanzar un objetivo concreto”.
- En la informática podríamos decir que es “la capacidad del producto software para ser entendido, aprendido, usado y resultar atractivo para el usuario”.
- El modelo conceptual de la usabilidad viene del **Diseño centrado en el usuario**
- Se encuentra definida en 2 normas ISO: 9126 y 9241

Usabilidad Web

- La usabilidad web, que parte del estudio del comportamiento del usuario, es la “capacidad de un producto para ser entendido, usado y ser atractivo en **cualquier dispositivo con conexión a internet**.
- Cuando diseñamos y programamos una web debemos pensar que el usuario necesita conseguir ese “objetivo” que nos marcamos cuando pensamos en el “producto”, si es capaz de conseguirlo, nuestra web podemos decir que es usable.

Diseño centrado en el usuario

- El Diseño Centrado en el Usuario (DCU), o User Centered Design (UCD), es definido por la *Usability Professionals Association* (UPA) como un enfoque de diseño cuyo proceso está dirigido por información sobre las personas que van a hacer uso del producto.
- El origen de esta visión se enmarca principalmente en el diseño industrial de la década de los 50.
- Los diseñadores estaban convencidos de que la adaptación y optimización al ser humano de diseño de productos respondía a un estudio minucioso de antropometría, ergonomía, arquitectura ó biomecánica.

Diseño centrado en el usuario

- En la Informática no llega este concepto hasta bien entrados los años 80, cuando una vez “superadas” limitaciones de procesamiento y almacenamiento comenzaron a interesarse por él.
- Fue el momento de observar como la gente usaba los sistemas y creaba sus modelos mentales a partir de los procesos de interacción.
- Hay 3 términos que deben valorarse para entender estos procesos:
 - El modelo conceptual: Ofrecido por el diseñador del sistema.
 - Interfaz: La imagen que el sistema presenta al usuario.
 - El modelo mental: Desarrollado por el usuario a partir de la imagen.

Diseño centrado en el usuario

- El objetivo de esta filosofía es ofrecer respuesta a preguntas como ¿quién usará este sistema?, ¿qué es lo que va a hacer con él? ó ¿qué información necesitará para alcanzar sus objetivos?
- Se habla como filosofía porque partimos de la premisa que condicionará todas nuestras acciones: **el usuario es el centro de toda decisión de diseño.**

Diseño centrado en el usuario

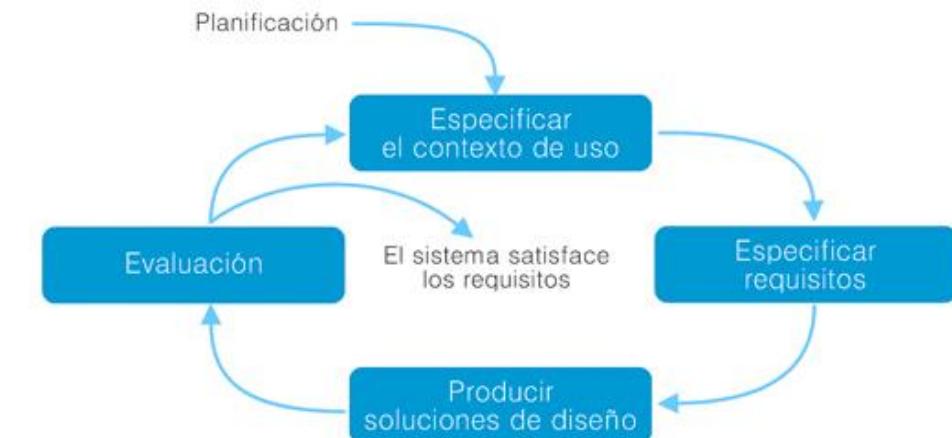
- Hay otros enfoques o filosofías de diseño:
 - **Diseño centrado en el diseñador** (Designer-centered design): El diseñador, a partir de su visión personal, sabe qué es lo mejor en cada momento.
 - **Diseño centrado en la empresa** (Enterprise-centered design): El sitio web se diseña atendiendo a la estructura y necesidades de la empresa.
 - **Diseño centrado en el contenido** (Content-centered design): El cuerpo de información es la base para organizar el sitio y la estructura de navegación.
 - **Diseño centrado en la tecnología** (Technology-centered design): Todo gira en torno a la tecnología y se busca la manera más fácil de implementar una solución.
- Afrontar un diseño desde este enfoque significa que debe ser el usuario final el que prevalezca sobre otros factores en la toma de decisiones, no que esos otros factores deban ser desatendidos

Diseño centrado en el usuario

- En ocasiones tendemos a confundir Usabilidad con DCU, aunque la usabilidad es un concepto inherente y central a la DCU, hay importantes diferencias en los conceptos.
- La usabilidad es un atributo de calidad del diseño, mientras que el DCU es una vía para alcanzar y mejorar empíricamente la usabilidad del producto.
- La DCU es un proceso cíclico en el que las decisiones de diseño están dirigidas por el usuario y los objetivos que pretende satisfacer el producto, y donde la usabilidad del diseño es evaluada de forma iterativa y mejorada incrementalmente.

Diseño centrado en el usuario

- Según la norma [ISO13407](#) el proceso tiene 4 partes:
 - Entender y especificar el contexto de uso: Identificar a las personas a las que se dirige el producto, para qué lo usarán y en qué condiciones.
 - Especificar requisitos: Identificar los objetivos del usuario y del proveedor del producto deberán satisfacerse.
 - Producir soluciones de diseño: Esta fase se puede subdividir en diferentes etapas secuenciales, desde las primeras soluciones conceptuales hasta la solución final de diseño.
 - Evaluación: Es la fase más importante del proceso, en la que se validan las soluciones de diseño (el sistema satisface los requisitos) o por el contrario se detectan problemas de usabilidad, normalmente a través de test con usuarios.



Reglas de la usabilidad web

- Existen 5 reglas:
 - **Tiempos de carga rápido:** cuanto más lenta cargue la web más aumenta la tasa de abandono y el porcentaje de rebote.
 - **Diseño limpio y claro:** para que el usuario pueda digerir de forma más sencilla la información de tu web. Es fundamental tener un diseño *responsive* y adaptar la web a dispositivos móviles, cada vez más usados.
 - **Accesibilidad:** cada vez son más webs, redes sociales y fabricantes que adaptan la tecnología teniendo en cuenta las dificultades físicas de los usuarios.

Reglas de la usabilidad web

- Coherencia: Es fundamental adaptar los campos de texto a las posibles respuestas, resaltar campos obligatorios, no solicitar información irrelevante, mostrar mensajes de confirmación y no hacer tedioso o largo el proceso de compra o inscripción.
- Claridad: Debes ponerte siempre en el lugar del usuario evitando conceptos complejos y aportar toda la información valiosa sobre tus productos y servicios.

Beneficios de la usabilidad

- Entre los beneficios se encuentran:
 - Reducción de los costes de aprendizaje y esfuerzos.
 - Disminución de los costes de asistencia y ayuda al usuario.
 - Disminución en la tasa de errores cometidos por el usuario y del retrabajo.
 - Optimización de los costes de diseño, rediseño y mantenimiento.

Beneficios de la usabilidad

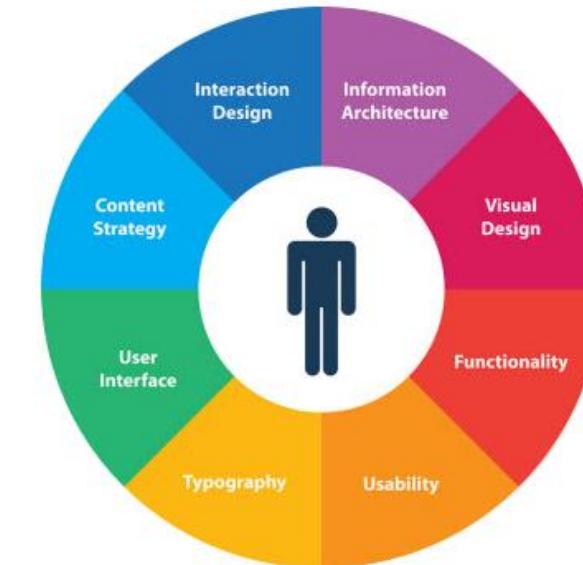
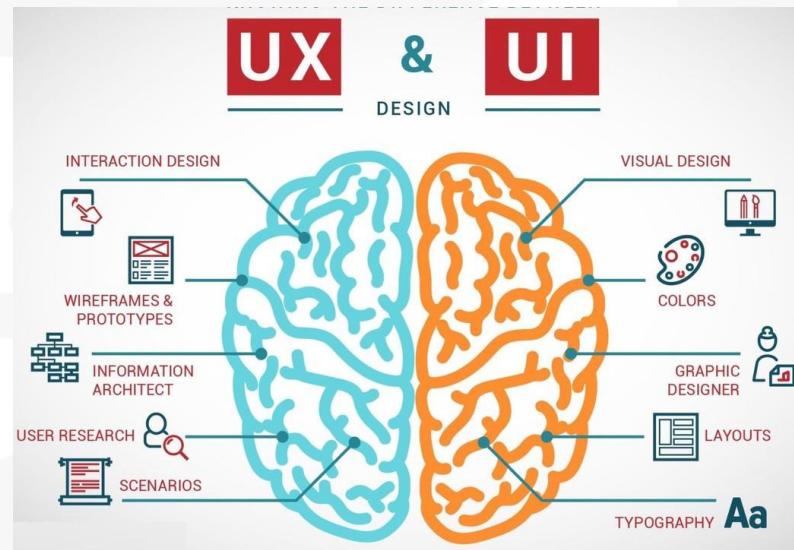
- Aumento de la tasa de conversión de visitantes a clientes de un sitio web.
- Aumento de la satisfacción y comodidad del usuario.
- Mejora la imagen y el prestigio.
- Mejora la calidad de vida de los usuarios, ya que reduce su estrés, incrementa la satisfacción y la productividad.

Como puedo mejorar la usabilidad

- Procura utilizar html lo mas estándar posible
- Usa CSS y JS en ficheros externos, sitúa siempre que puedas los ficheros JS al final del documento
- Agrupa y comprime los ficheros de código
- Carga el menor numero de elementos multimedia posibles
- Comprime fotos
- No almacenes las imágenes en servidores de terceros
- Indica los datos de altura y anchura de las imágenes.

Experiencia de usuario

- La experiencia de usuario (UX) es un atributo de calidad que mide la facilidad de uso de las interfaces.
- No es mas que el resultado de las percepciones y respuestas de las personas por el uso de un producto, sistema o servicio.



Como generamos UX

- **Usabilidad:** es muy importante que el sitio web sea eficiente para que los visitantes puedan cumplir el objetivo que pretendían al acceder a la plataforma.
- **Utilidad:** todos los elementos de la web deben tener un propósito concreto. La revisión debe ser constante para mejorar.
- **Confianza:** es importante establecer explicaciones a modo de FAQ para que el usuario disponga de toda la información necesaria en cada paso.
- **Deseo:** El diseño, la estética y un contenido multimedia de calidad pueden hacer mucho más deseable un producto, incidiendo directamente en una experiencia positiva de usuario.
- **Facilidad de búsqueda:** El buscador, la nube de etiquetas y las secciones y subsecciones deben responder a las necesidades de consumidores e internautas a fin de que el consumidor quede satisfecho y tenga una experiencia positiva, encontrando en pocos segundos lo que estaba buscando.
- **Seguridad:** fundamental en lo que concierne a los datos personales o información confidencial.
- **Creación de valor:** La información de la web o los productos que se venden deben tener un valor más allá de lo económico, llenando las expectativas de los usuarios y diferenciándola de tu competencia.

Accesibilidad

- “Condición que deben cumplir los entornos, procesos, bienes, productos y servicios, así como los objetos o instrumentos, herramientas y dispositivos, para ser comprensibles, utilizables y practicables por todas las personas en condiciones de seguridad y comodidad y de la forma más autónoma y natural posible.”



Accesibilidad

- La accesibilidad en la web también está contemplado por el W3C
- Está regido por la Web Accessibility Initiative (WAI) que ha ido proporcionando guías para la programación y adaptación de la web.
- En estos momentos la normativa está en su versión WCAG 2.1 y próxima a salir la siguiente revisión.
- Esta normativa se basa en una serie de principios y pautas.

Principios de la accesibilidad

- Hay 4 principios básicos en la accesibilidad web:
 - Perceptible: la información y los componentes de la interfaz de usuario deben ser mostrados a los usuarios en formas que ellos puedan entender
 - Operable: Los componentes de la interfaz de usuario y la navegación debe ser manejable
 - Comprensible: La información y las operaciones de usuarios deben ser comprensibles
 - Robusto: el contenido deber ser suficientemente robusto para que pueda ser bien interpretado por una gran variedad de agentes de usuario, incluyendo tecnologías de asistencia



Pautas de accesibilidad - Perceptible

- Pauta 1.1: Texto alternativo: Proporciona texto alternativo para el contenido que no sea textual, así podrá ser transformado en otros formatos que la gente necesite, como caracteres grandes, lenguaje braille, lenguaje oral, símbolos o lenguaje más simple.
- Pauta 1.2: Contenido multimedia dependiente del tiempo: Proporcione alternativas sincronizadas para contenidos multimedia sincronizados dependientes del tiempo.
- Pauta 1.3: Adaptable: Crear contenido que pueda ser presentado de diferentes formas sin perder ni información ni estructura.
- Pauta 1.4: Distinguible: Facilitar a los usuarios ver y escuchar el contenido incluyendo la distinción entre lo más y menos importante.

Pautas de accesibilidad - Operable

- Pauta 2.1: Teclado accesible: Poder controlar todas las funciones desde el teclado.
- Pauta 2.2 Tiempo suficiente: Proporciona tiempo suficiente a los usuarios para leer y utilizar el contenido.
- Pauta 2.3: Ataques epilépticos: No diseñar contenido que pueda causar ataques epilépticos.
- Pauta 2.4: Navegación: Proporciona formas para ayudar a los usuarios a navegar, a buscar contenido y a determinar donde están estos.
- Pauta 2.5: Modalidades de entrada: Facilitar a los usuarios operar la funcionalidad a través de varios métodos de entrada además del teclado.

Pautas de accesibilidad - Comprendible

- Pauta 3.1: Legible. Hacer contenido de texto legible y comprensible.
- Pauta 3.2 Previsible: Hacer la apariencia y la forma de utilizar las páginas web previsibles.
- Pauta 3.3 Asistencia a la entrada de datos: utilizar ayudas que evitarán y corregirán errores.

Pautas de accesibilidad – Robustez

- Pauta 4.1 Compatible: Maximiza la compatibilidad con los agentes de usuario actuales y futuros, incluyendo tecnologías de asistencia.
 - Tecnologías de asistencia: lectores en pantalla, línea braille, herramientas de magnificación de pantallas, entradas alternativas a ratón y teclado tradicionales.



Criterios de cumplimiento de la accesibilidad

- Hay 3 niveles de conformidad: A, AA, AAA
- Tiene que aplicarse a la página web COMPLETA
- Se aplica también a los procesos COMPLETOS
- Sin interferencias externas (en caso de que haya motivos para no tener una parte accesible, se dará una alternativa)

Evaluación de la accesibilidad

- Título de la página
- Textos alternativos para las imágenes
- Uso de cabeceras, contraste de colores y texto redimensionable
- Acceso desde el teclado
- Formularios, etiquetas y errores accesibles.
- Eliminar movimientos, flashes o contenido parpadeante
- Alternativas accesibles a video y audio.
- Estructura de la página correcta.

Herramientas para la evaluacion de la accesibilidad

- No hay una única herramienta para evaluar la accesibilidad de las páginas web
- Podemos encontrar una extensa lista de ellas en la página de WAI del W3C

<https://www.w3.org/WAI/ER/tools/?q=wcag-20-w3c-web-content-accessibility-guidelines-20>

Seguridad en aplicaciones Web

Javier Albert Segui
Curso 2020/21

Seguridad Web

- La seguridad web es la acción/práctica de proteger sitios web del acceso, uso, modificación, destrucción o interrupción, no autorizados.
- La seguridad de sitios web eficaz requiere de esfuerzos de diseño a lo largo de la totalidad del sitio web: en tu aplicación web, en la configuración del servidor web, en tus políticas para crear y renovar contraseñas, y en el código del lado cliente.
- Debemos seguir siempre la siguiente premisa

Planear – Hacer – Verificar – Actuar

Aplicaciones seguras

- Las aplicaciones seguras no se dan por si mismas, son el resultado de un esfuerzo en todos los implicados en las aplicaciones.
- Para conseguir una aplicación segura necesitamos:
 - Una organización que abogue por la seguridad
 - Políticas de seguridad documentadas y basadas en estándares
 - Metodologías de desarrollo con puntos de control y actividades de seguridad
 - Gestión segura de versiones y configuraciones.

Metodologías de desarrollo

- La elección de una u otra metodología no es tan importante como el hecho de poseer una.
- Debemos buscar las siguientes características:
 - Fuerte aceptación de las fases de diseño, testeo y documentación.
 - Espacios donde poder insertar controles de seguridad (Análisis de riesgos, amenazas, revisiones de código....)
 - Que sea funcional para el tamaño de la organización
 - Tenga potencial para reducir la tasa de errores y mejorar la productividad de los desarrolladores

Estándares de codificación

- Una metodología no es un estándar de codificación
- Deberíamos considerar:
 - Orientación de la arquitectura
 - Niveles mínimos de documentación
 - Requisitos de testeo obligatorios
 - Niveles mínimos de comentarios
 - Uso de los bloques de control
 - Método de nombrado de variables, métodos, clases y tablas.
 - Código lo mas legible posible antes que código complejo e “ideas felices”
 - Control del código fuente

Seguridad de la información

- La seguridad de la información se basa en 3 pilares que son:
 - **Confidencialidad:** permitir acceso únicamente a los datos a los cuales el usuario tiene permitido
 - **Integridad:** Asegurar que los datos no se falsifican o alteran por usuarios no autorizados
 - **Disponibilidad:** Asegurar que los sistemas y los datos están disponibles para los usuarios autorizados cuando lo necesitan

Principios de la codificación segura

- Minimizar la superficie de ataque
- Seguridad por defecto
- Principio del mínimo privilegio
- Principio de la defensa en profundidad
- Fallos de manera segura
- Separación de funciones
- Simplicidad
- Parches de seguridad

Pruebas

- Las pruebas o testing son un proceso de comparación del estado de algo ante un conjunto de criterios.
- Muchas veces este conjunto de criterios es difuso y no está estandarizado o escrito en un documento de pruebas.
- Las pruebas no son una fase adicional al desarrollo de software, sino que deben estar inmersas durante todo el ciclo de vida del mismo.
- Utiliza las herramientas adecuadas.
- Desarrolla métricas
- Por tanto, debemos tener la máxima de “Prueba pronto, prueba a menudo”

Técnicas de pruebas

- Inspecciones y revisiones manuales

- Las inspecciones manuales son revisiones realizadas por personas, que por lo general comprueban las implicaciones de seguridad de personas, políticas y procesos, aunque pueden incluir la inspección de decisiones tecnológicas, como puede ser los diseños de la arquitectura escogidos

- Modelado de amenazas

- es una técnica popular para ayudar a los diseñadores de sistemas acerca de las amenazas de seguridad a las que se enfrentan sus sistemas. Les permite desarrollar estrategias de mitigación para vulnerabilidades potenciales. El modelado de amenazas ayuda a las personas a concentrar sus, inevitablemente, limitados recursos y atención en aquellas partes del sistema que más lo necesitan

Técnicas de pruebas

- Revisión de código

- es el proceso de comprobar manualmente el código fuente de una aplicación web en busca de incidencias de seguridad.

- Pruebas de intrusión

- Las pruebas de intrusión son esencialmente el “arte” de comprobar una aplicación en ejecución remota, sin saber el funcionamiento interno de la aplicación, para encontrar vulnerabilidades de seguridad

Top 10 de Riesgos en las aplicaciones Web

- **Inyección:** Las fallas de inyección, como SQL, NoSQL, OS o LDAP ocurren cuando se envían datos no confiables a un intérprete, como parte de un comando o consulta. Los datos dañinos del atacante pueden engañar al intérprete para que ejecute comandos involuntarios o acceda a los datos sin la debida autorización.
- **Perdida de autenticación:** Las funciones de la aplicación relacionadas a autenticación y gestión de sesiones son implementadas incorrectamente, permitiendo a los atacantes comprometer usuarios y contraseñas, token de sesiones, o explotar otras fallas de implementación para asumir la identidad de otros usuarios (temporal o permanentemente).
- **Exposición de datos sensibles:** Muchas aplicaciones web y APIs no protegen adecuadamente datos sensibles, tales como información financiera, de salud o Información Personalmente Identificable (PII). Los datos sensibles requieren métodos de protección adicionales, como el cifrado en almacenamiento y tránsito

Top 10 de Riesgos en las aplicaciones Web

- **Entidades Externas XML:** Muchos procesadores XML antiguos o mal configurados evalúan referencias a entidades externas en documentos XML.
- **Perdida de control de acceso:** Las restricciones sobre lo que los usuarios autenticados pueden hacer no se aplican correctamente. Los atacantes pueden explotar estos defectos para acceder, de forma no autorizada, a funcionalidades y/o datos.
- **Configuración de Seguridad Incorrecta:** La configuración de seguridad incorrecta es un problema muy común y se debe en parte a establecer la configuración de forma manual, ad hoc o por omisión (o directamente por la falta de configuración).

Top 10 de Riesgos en las aplicaciones Web

- Cross Site Scripting (XSS): es un tipo de vulnerabilidad informática o agujero de seguridad típico de las aplicaciones Web, que puede permitir a una tercera persona injectar en páginas web visitadas por el usuario código JavaScript o en otro lenguaje similar
- Deserialización Insegura: Estos defectos ocurren cuando una aplicación recibe objetos serializados dañinos y estos objetos pueden ser manipulados o borrados por el atacante para realizar ataques de repetición, inyecciones o elevar sus privilegios de ejecución

Top 10 de Riesgos en las aplicaciones Web

- Componentes con vulnerabilidades conocidas: Los componentes como bibliotecas, frameworks y otros módulos se ejecutan con los mismos privilegios que la aplicación.
- Registro y Monitorización Insuficientes: El registro y monitoreo insuficiente, junto a la falta de respuesta ante incidentes permiten a los atacantes mantener el ataque en el tiempo, pivotear a otros sistemas y manipular, extraer o destruir datos.

Recursos

o OWASP

- <https://owasp.org/>
- Top Ten: <https://owasp.org/www-project-top-ten/>
- Testing Guide: <https://owasp.org/www-project-web-security-testing-guide/>
- Development Guide: https://owasp.org/www-pdf-archive/Owasp_Dev_Guide.pdf