Unidad 1:
Conceptos generales sobre servicios web

Índice

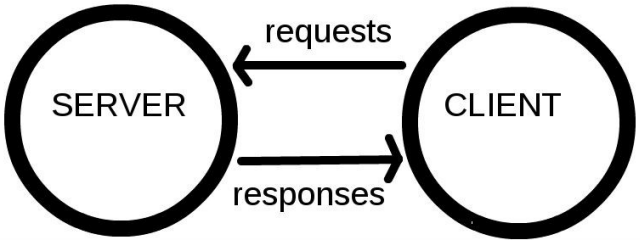
Unidad 1 Conceptos generales sobre servicios web....1
1. Concepto de cliente y de servidor....2
1.1. Los clientes y servidores....2
1.2. Explicando los DNS....3
2. Programación de Lado-Servidor y del Lado-Cliente....4
2.1. ¿Son iguales las programaciones lado-servidor y lado-cliente?....4
2.2. ¿Qué se puede hacer en el lado-servidor?....4
Almacenaje y distribución eficiente de información....4
Experiencia de usuario personalizada....5
Acceso controlado al contenido....5
Almacenar información de sesión/estado....5
Notificaciones y comunicación....5
Análisis de datos....6
3. Sitios estáticos y sitios dinámicos....6
3.1. Sitios estáticos....6
3.2. Sitios dinámicos....7
4. Descripción de la plataforma Java, Enterprise Edition....9
4.1. Aplicaciones escalonadas....10
5. Servidores y servidores Java EE....11
6. Tipos de aplicaciones web....12
6.1. Aplicaciones web en capa de presentación....12
6.2. Aplicaciones web orientadas a servicios....12
7. Concepto de servicio web....13
7.1. Ventajas de los servicios web....14
8. Arquitectura orientada a servicios (SOA, siglas del inglés *Service Oriented Architecture*)....15
8.1. Servicios Web SOAP....15
8.2. Servicios Web REST....16
8.3. Diferencias con SOAP....17

1. Concepto de cliente y de servidor

1.1. Los clientes y servidores

Los ordenadores conectados a la Web pueden ser **clientes** y/o **servidores**.

Un diagrama simplificado de cómo interactúan se vería así:



- Los **clientes** son dispositivos conectados a Internet (por ejemplo, tu ordenador conectado a la red Wi-Fi o el teléfono conectado a la red de telefonía móvil) y el software que se encuentra disponible y permite acceder a Internet en dichos dispositivos (normalmente, un navegador web como Firefox, Chrome, etc).
- Los **servidores** son ordenadores que almacenan páginas web, sitios web o aplicaciones. Cuando un dispositivo cliente quiere acceder a una página web, una página web se descarga desde el servidor en el equipo cliente y se mostrará en el navegador web del usuario.
- **Tu conexión a Internet**: Permite enviar y recibir datos en la Web.
- **TCP/IP**: **Protocolo de Control de Transmisión** y **Protocolo de Internet**, son los protocolos de comunicación que definen cómo deben viajar los datos a través de la Web. Esto es, los medios de transporte que te permiten hacer un pedido, ir a la tienda y comprar los productos.
- **DNS:** Los servidores del **Sistema de Nombres de Dominio** (DNS), son como una libreta de direcciones de sitios web. Cuando escribes una dirección web en el navegador, el navegador busca los DNS antes de recuperar el sitio web. El navegador necesita averiguar en qué servidor vive el sitio web y así enviar los mensajes HTTP al lugar correcto.
- **HTTP:** El **Protocolo de Transferencia de Hipertexto** es un protocolo de aplicación que define un idioma para que los clientes y servidores se puedan comunicar.
- **Archivos componentes**: Un sitio web se compone de muchos archivos diferentes. Estos archivos se dividen en dos tipos principales:
    - **Archivos de código**: los sitios web se construyen principalmente con HTML, CSS y JavaScript, aunque te encontrarás con otras tecnologías más adelante.

- **Recursos:** Este es un nombre colectivo para el resto de materiales que conforman un sitio web, como imágenes, música, video, documentos de Word, archivos PDF, etc.

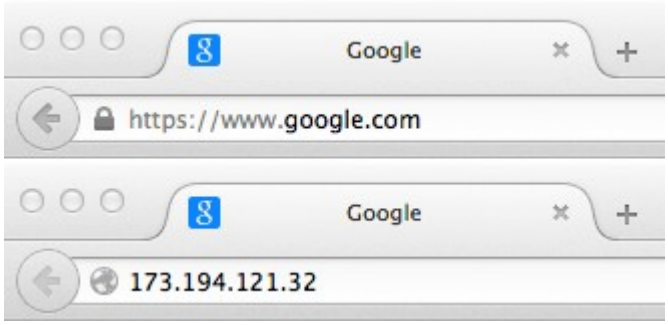
Cuando escribes una dirección web en el navegador (usando una analogía para realizar venta on line):

1. El navegador va al servidor DNS y encuentra la dirección real del servidor donde el sitio web vive (encontrar la dirección de la tienda).
2. El navegador envía un mensaje de petición HTTP al servidor, pidiéndole que envíe una copia de la página web para el cliente (ir a la tienda y hacer un pedido). Este mensaje y todos los datos enviados entre el cliente y el servidor, se envían a través de tu conexión a Internet usando TCP/IP.
3. Siempre que el servidor apruebe la solicitud del cliente, el servidor enviará al cliente un mensaje "200 OK", que significa, "¡por supuesto que puedes ver ese sitio web! Aquí está.", y comenzará a enviar los archivos de la página web al navegador como una serie de pequeños trozos llamados paquetes de datos.
4. El navegador reúne los pequeños trozos, forma un sitio web completo y te lo muestra .

1.2. Explicando los DNS

Una dirección web se identifica de forma única mediante una **dirección IP.**

Los servidores de nombres de dominio hacen coincidir una dirección web tecleada desde tu navegador ("mozilla.org", por ejemplo) con la dirección real del sitio web (IP).



Los datos se envían a través de la Web como miles de trozos pequeños, permitiendo que muchos usuarios pueden descargar la misma página web al mismo tiempo. Si los sitios web fueran enviados como grandes trozos, sólo un usuario podría descargarlos a la vez, lo que volvería a la Web muy ineficiente.

2. Programación de Lado-Servidor y del Lado-Cliente

2.1. ¿Son iguales las programaciones lado-servidor y lado-cliente?

Tienen diferentes propósitos y preocupaciones.

- Normalmente no usan los mismos lenguajes de programación (siendo la excepción el JavaScript, que puede usarse tanto en lado servidor como en lado cliente).
- Se pueden ejecutar en entornos de diferentes sistemas operativos.
- El código que se ejecuta en el explorador se conoce como **código de lado-cliente**, y su principal preocupación es la mejora de la apariencia y el comportamiento de una página web entregada. Esto incluye la selección y estilo de los componentes UI, la creación de diseños, navegación, validación de formularios, etc.
- Por contra, la programación de sitios web de lado servidor en su mayor parte implica la elección de *qué contenido* se ha de devolver al explorador como respuesta a sus peticiones. El código de lado-servidor gestiona tareas como la validación de los datos enviados y las peticiones, usando bases de datos para almacenar y recuperar datos, y enviando los datos correctos al cliente según se requiera.
- El código del lado cliente está escrito usando HTML, CSS, y JavaScript — es ejecutado dentro del explorador web y tiene poco o ningún acceso al sistema operativo subyacente (incluyendo un acceso limitado al sistema de ficheros).
- El código del lado servidor puede escribirse en cualquier número de lenguajes de programación — ejemplos de lenguajes de programación populares incluyen Java, PHP, Python, Ruby, C#. El código del lado servidor tiene acceso completo al sistema operativo del servidor y el desarrollador puede elegir qué lenguaje de programación (y qué versión específica) desea usar.

2.2. ¿Qué se puede hacer en el lado-servidor?

- **Almacenaje y distribución eficiente de información**

Imagina cuántos productos están disponibles en Amazon, e imagina cuántas entradas se han escrito en Facebook. Crear una página estática separada para cada producto o entrada sería completamente ineficiente.
La programación de lado-servidor nos permite por el contrario almacenar la información en una base de datos y construir dinámicamente y devolver ficheros HTML y de otros tipos (ej, PDFs, imágenes, etc.). También es posible devolver simplemente datos (JSON, XML, etc.)
Debido a que la información está en una base de datos, puede también ser compartida y actualizada con otros sistemas de negocio (por ejemplo, cuando se venden los productos online o en una tienda, la tienda debería actualizar su base de datos de inventario.

▪ Experiencia de usuario personalizada

Los servidores pueden almacenar y usar la información acerca de los clientes para proporcionar una experiencia de usuario conveniente y dirigida. Por ejemplo, muchos usuarios almacenan tarjetas de crédito de forma que los detalles no tienen que ser introducidos de nuevo. Sitios como Google Maps usan la localización de tu casa y la actual para proporcionar una información sobre la ruta a seguir y resaltar los negocios locales en los resultados de búsqueda.
Un análisis profundo de los hábitos del usuario se puede usar para anticipar sus intereses y personalizar las respuestas y notificaciones futuras, proporcionando, por ejemplo, una lista de las localizaciones visitadas o populares que querrías buscar en un mapa.

▪ Acceso controlado al contenido

La programación de lado-servidor permite a los sitios restringir el acceso a usuarios autorizados y servir sólo la información que se le permite ver al usuario.
Ejemplos del mundo real incluyen:

- Redes sociales como Facebook permiten a los usuarios controlar totalmente sus propios datos pero permitiendo sólo a sus amigos ver o comentar sobre ellos.
- Acceso a cuentas de banco.

• Almacenar información de sesión/estado

La programación de lado-servidor permite a los desarrolladores hacer uso de las **sesiones** — es básicamente un mecanismo que permite al servidor almacenar información sobre el usuario actual del sitio o enviar diferentes respuestas basadas en esa información. Esto permite, por ejemplo, que un sitio sepa que un usuario ha iniciado sesión previamente y presente enlaces a sus correos, o a su historial de órdenes, o quizá guardar el estado de un simple juego de forma que el usuario pueda volver al sitio de nuevo y retomar el juego donde lo dejó.

▪ Notificaciones y comunicación

Los servidores pueden enviar notificaciones de tipo general o específicas de usuario a través del propio sitio web o vía correo electrónico, SMS, mensajería instantánea, conversaciones de video u otros servicios de comunicación.
Unos pocos ejemplos incluyen:

- Facebook y Twitter envían mensajes de correo y SMS para notificarte de nuevas comunicaciones.
- Amazon envía con regularidad emails que sugieren productos similares a aquellos comprados o vistos anteriormente y en los que podrías estar interesado.
- Un servidor web podría enviar mensajes de aviso a los administradores del sistema alertandoles de memoria baja en el servidor o de actividades de usuario sospechosas.

Nota: El tipo de notificación más común es una "confirmación de registro". Elige uno cualquiera de los grandes sitios en que estés interesado (Google, Amazon, Instagram, etc.) y crea una cuenta nueva usando tu dirección de correo. En breve recibirás un email de confirmación de registro, o solicitando un acuse de recibo para activar la cuenta.

- **Análisis de datos**

Un sitio web puede recolectar un montón de datos acerca de los usuarios: qué es lo que buscan, qué compran, qué recomiendan, cuánto tiempo permanecen en cada página. La programación de lado-servidor puede utilizarse para refinar las respuestas basándose en el análisis de estos datos.
Por ejemplo, Amazon y Google anuncian ambos productos basados en búsquedas previas (y adquisiciones).

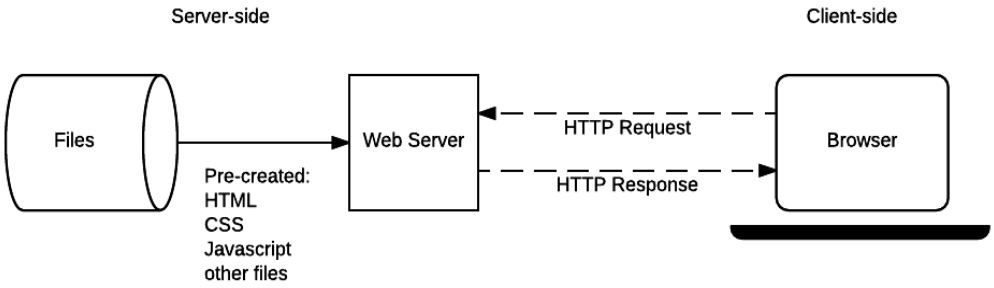
Hemos aprendido que el código de lado-servidor se ejecuta en un servidor web y que su papel principal es controlar *qué* información se envía al usuario (mientras que el código de lado-cliente gestiona principalmente la estructura y presentación de esos datos al usuario).

3. Sitios estáticos y sitios dinámicos

3.1. Sitios estáticos

Un *sitio estático* es aquél que devuelve desde el servidor el mismo contenido establecido de forma fija en el código cada vez que se solicita una página en particular. De manera que si por ejemplo tienes una página sobre un producto en `/static/myproduct1.html` , a todos los usuarios se les devolverá la misma página. Si añades otro producto similar a tu sitio necesitarás añadir otra página (ej. `myproduct2.html`) etc... Esto puede llegar a ser realmente muy poco eficiente — ¿qué sucede cuando alcanzas miles de páginas de productos? Repetirías un montón de código a lo largo de cada página (la plantilla básica de la página, la estructura, etc), y si quisieras cambiar cualquier cosa de la estructura de la página — como añadir una nueva sección de "productos relacionados" por ejemplo — tendrías que cambiar cada página individualmente.
Nota: Los sitios estáticos son excelentes cuando tienes un pequeño número de páginas y quieres enviar el mismo contenido a todos los usuarios. Sin embargo pueden tener un coste de mantenimiento significante a medida que es número de páginas se hace grande.

El diagrama de la arquitectura de un sitio estático sería:



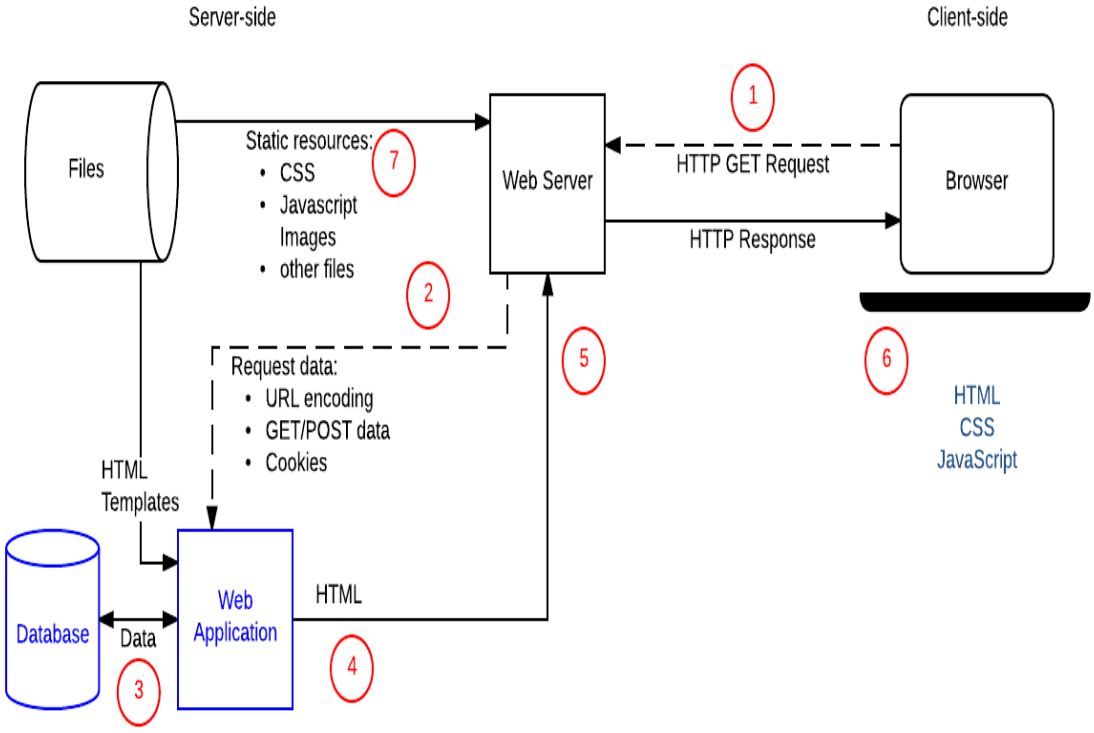
Cuando un usuario quiere navegar a una página, el explorador envía una petición HTTP `GET` especificando la URL de su página HTML. El servidor recupera el documento solicitado de su sistema de ficheros y devuelve una respuesta HTTP conteniendo el documento y un código de estado de respuesta HTTP `"200 OK"` (indicando éxito). El servidor podría devolver un código de estado diferente, por ejemplo `"404 Not Found"` si el fichero no está presente en el servidor, o `"301 Moved Permanently"` si el fichero existe pero ha sido redirigido a una localización diferente.
El servidor de un sitio estático sólo tendrá que procesar peticiones GET, ya que el servidor no almacena ningún dato modificable. Tampoco cambia sus respuestas basándose en los datos de la petición HTTP (ej. parámetros URL o cookies).

3.2. Sitios dinámicos

Un *sitio dinámico* es aquél que puede generar y devolver contenido basándose en la URL y los datos específicos de la petición (en vez de devolver siempre para una URL en particular el mismo fichero especificado en el código de forma fija). Usando el ejemplo de un sitio de productos, el servidor almacenaría "datos" del producto en una base de datos en vez de ficheros HTML individuales. Cuando se reciba una petición HTTP `GET` para un producto, el servidor determina el ID del mismo, extrae los datos de la base y construye la página HTML de la respuesta insertando los datos dentro de la plantilla HTML. Esto tiene una ventaja primordial sobre un sitio estático:
Usar una base de datos permite que la información del producto se almacene de forma eficiente y que se pueda ampliar, modificar y buscar fácilmente.

Un sitio dinámico es aquél en que algún contenido de la respuesta está generado *dinámicamente* sólo cuando se necesita. En un sitio web dinámico las páginas HTML se crean normalmente insertando datos desde una base en variables dentro de plantillas HTML (esta es una forma mucho más eficiente de almacenar gran cantidad de contenido que la que usan los sitios web estáticos). Un sitio dinámico puede devolver datos diferentes para un URL basados en la información proporcionada por el usuario o sus preferencias almacenadas y puede realizar otras operaciones como parte de la devolución de respuesta (ej, enviar notificaciones).
La mayor parte del código para soportar un sitio web dinámico debe correr en el servidor. La creación de este código se conoce como "**programación de lado-servidor**" (o algunas veces "**back-end scripting**").

El diagrama de abajo muestra una arquitectura simple para un *sitio web dinámico*. Como en el diagrama previo, los navegadores web envían peticiones HTTP al servidor, el servidor procesa a continuación las peticiones y devuelve las respuestas HTTP apropiadas. Las peticiones de recursos *estáticos* son gestionadas de la misma manera que para los *sitios estáticos* (los recursos estáticos son cualquier fichero que no cambia - generalmente: CSS, JavaScript, Imágenes, ficheros PDF creados previamente, etc...)



Las peticiones de recursos dinámicos, por el contrario, son reenviadas (2) al código del lado-servidor (mostrado en el diagrama como *Web Application*). Para las "peticiones dinámicas" el servidor interpreta la petición, lee de la base de datos la información requerida (3), combina los datos recuperados con las plantillas HTML (4), y envía de vuelta una respuesta que contiene el HTML generado (5,6).

1. El navegador web crea una petición HTTP `GET` al servidor usando la URL base del recurso (`/best`) y codifica el equipo y número de jugadores como parámetros URL (ej. `/best?team=my_team_name&show=11`) o formando parte de un patrón

URL (ej. `/best/my_team_name/11/`). Se usa una petición `GET` porque la petición sólo recoge datos (no modifica ninguno).
2. El S*ervidor Web* detecta que la petición es "dinámica" y la reenvía a la *Aplicación* para que la procese (el servidor web determina como manejar diferentes URLs basándose en reglas de emparejamiento de patrones definidas en su configuración).
3. La *Aplicación Web* identifica la intención de la petición y obtiene la información solicitada de la base de datos.
4. La *Aplicación Web* crea dinámicamente una página HTML respuesta.
5. La *Aplicación Web* devuelve el HTML generado al explorador web (via el *Servidor Web*), junto con un código de estado HTTP de 200 ("éxito"). Si algo impide que se pueda devolver el HTML entonces la *Aplicación Web* devolverá otro código — por ejemplo "404" para indicar que la información requerida no existe.
6. El Explorador Web comenzará a continuación a procesar el HTML devuelto, enviando peticiones separadas para obtener cualquier otro fichero CSS o JavaScript que sea referenciado (ver paso 7).
7. El Servidor Web carga ficheros estáticos del sistema de ficheros y los devuelve al explorador directamente (de nuevo, la gestión correcta de los ficheros está basada en las reglas de configuración y de emparejamiento de patrones URL).

La operación de actualizar un registro de la base de datos se gestionaría de forma similar, excepto que, como para cualquier actualización de la base de datos, la petición HTTP desde el explorador debería ser codificada como petición `POST`.
Una aplicación web dinámica puede también tareas que no impliquen acceso a datos, por ejemplo realizar logging, etc.
El código lado servidor de un sitio web no tiene que devolver fragmentos/ficheros HTML en la respuesta. Puede en vez de eso crear dinámicamente y devolver otros tipos de ficheros (texto, PDF, CSV, etc.) o incluso datos (jSON, XML, etc.).

4. Descripción de la plataforma Java, Enterprise Edition

El objetivo de la plataforma Java EE es proporcionar a los desarrolladores un conjunto potente de API, acortando el tiempo de desarrollo, reduciendo la complejidad de las aplicaciones y mejorando el rendimiento de las aplicaciones.

Las características que hacen que las aplicaciones empresariales sean potentes, como la seguridad y la confiabilidad, a menudo hacen que estas aplicaciones sean complejas. La plataforma Java EE reduce la complejidad del desarrollo de aplicaciones empresariales al proporcionar un modelo de desarrollo, API y entorno de tiempo de ejecución que permite a los desarrolladores concentrarse en la funcionalidad.

4.1. Aplicaciones escalonadas

En una aplicación de varios niveles, la funcionalidad de la aplicación está separada en áreas funcionales aisladas, llamadas niveles.

Por lo general, las aplicaciones de varios niveles tienen:

- un nivel de cliente (consiste en un programa de cliente que realiza solicitudes al nivel medio),
- un nivel medio (que se divide en un nivel web y un nivel empresarial, que manejan las solicitudes de los clientes y procesan los datos de las aplicaciones).
- un nivel de datos (a menudo denominado nivel de sistemas de información empresarial).

El desarrollo de aplicaciones Java EE se concentra en el nivel intermedio para hacer que la administración de aplicaciones empresariales sea más fácil, más robusta y más segura.

El nivel del cliente

El nivel de cliente consiste en clientes de aplicaciones que acceden a un servidor Java EE y que generalmente se encuentran en una máquina diferente del servidor. Los clientes hacen solicitudes al servidor. El servidor procesa las solicitudes y devuelve una respuesta al cliente. Muchos tipos diferentes de aplicaciones pueden ser clientes Java EE, y no siempre son, o incluso a menudo, aplicaciones Java. Los clientes pueden ser un navegador web, una aplicación independiente u otros servidores, y se ejecutan en una máquina diferente del servidor Java EE.

El nivel web

El nivel web consta de componentes que manejan la interacción entre los clientes y el nivel empresarial. Sus tareas principales son las siguientes:

- Generar dinámicamente contenido en varios formatos para el cliente.
- Recopilar información de los usuarios de la interfaz del cliente y devuelva los resultados apropiados de los componentes en el nivel empresarial.
- Controlar el flujo de pantallas o páginas en el cliente.
- Mantener el estado de los datos para la sesión de un usuario.
- Realice una lógica básica y mantenga algunos datos temporalmente en beans administrados.

El Nivel de Negocio o empresarial

El nivel empresarial consta de componentes que proporcionan la lógica comercial para una aplicación. La lógica empresarial es un código que proporciona funcionalidad a un dominio comercial particular, como la industria financiera o un sitio de comercio electrónico. En una aplicación empresarial diseñada correctamente, la funcionalidad principal existe en los componentes de nivel empresarial.

Las siguientes tecnologías Java EE se encuentran entre las que se utilizan en el nivel empresarial en las aplicaciones Java EE:

- Componentes de Enterprise JavaBeans (enterprise bean)
- Servicios web RESTful de JAX-RS
- Entidades Java Persistence API

El nivel de datos o de los sistemas de información empresarial

El nivel de sistemas de información empresarial (EIS) consta de servidores de bases de datos, sistemas de planificación de recursos empresariales y otras fuentes de datos heredadas, como los mainframes. Por lo general, estos recursos se encuentran en una máquina separada del servidor Java EE, y los componentes del nivel empresarial acceden a ellos.

Las siguientes tecnologías Java EE se utilizan para acceder al nivel EIS en aplicaciones Java EE:

- La API de conectividad de la base de datos Java (JDBC)
- La API de Java Persistence
- La arquitectura del conector EE de Java
- La API de transacción de Java (JTA)

5. Servidores y servidores Java EE

Un servidor Java EE es una aplicación de servidor que implementa las API de plataforma Java EE y proporciona servicios Java EE estándar. Los servidores Java EE a veces se denominan servidores de aplicaciones, porque le permiten enviar datos de la aplicación a

los clientes, del mismo modo que los servidores web sirven las páginas web para los navegadores web.

Los servidores Java EE alojan varios tipos de componentes de aplicación que corresponden a los niveles en una aplicación de varios niveles. El servidor Java EE proporciona servicios a estos componentes en forma de contenedor.

6. Tipos de aplicaciones web

En JEE podemos crear dos tipos de aplicaciones web:

- **Aplicaciones web orientadas en capa de presentación.**
- **Aplicaciones web orientadas a servicios.**

6.1. Aplicaciones web en capa de presentación

La capa de presentación es la parte de nuestra aplicación empresarial que interactúa directamente con los usuarios. A través de la capa de presentación se intercambian se introducen datos de entrada, se consultan datos de salida o se eligen diferentes opciones de proceso en un menú.
En aplicaciones web, la capa de presentación es construida básicamente en HTML.
JEE tiene las tecnologías JSF para desarrollar aplicaciones web en capa de presentación.

6.2. Aplicaciones web orientadas a servicios

Las aplicaciones web orientadas a servicios están diseñadas para intercambiar datos entre diferentes aplicaciones en vez de presentar una interfaz para interactuar directamente con el usuario final.

Los servicios web permiten que aplicaciones escritas en diferentes lenguajes de programación y ejecutándose en diferentes sistemas operativos y en diferentes plataformas de hardware puedan intercambiar información entre ordenadores interconectados entre sí por una red.

JEE posee tecnologías que soportan la creación de servicios web como **JAX-WS** o **JAX-RS**

7. Concepto de servicio web

Un problema que ocurre a menudo es el siguiente: ¿cómo se puede conseguir que aplicaciones escritas en distintos lenguajes intercambien datos entre sí?

Supongamos que en un departamento de una empresa se dispone de una aplicación escrita en C# a la que se le puede suministrar un nombre de mes y que devuelve los mejores clientes de ese mes. Y supongamos por otro lado que se dispone de una aplicación Java de escritorio que permite a los clientes mostrar gráficos de una manera muy cómoda. ¿Como conseguir que la aplicación Java se integre con la aplicación en C#?

- Una posibilidad sería reescribir la funcionalidad. Tomar la tarea hecha en C# y reescribirla en la aplicación Java. Evidentemente, esto llevaría bastante tiempo.
- Otra posibilidad sería inventar algún protocolo que permitiese llevar los datos entre distintas aplicaciones, lo cual también puede ser farragoso y propenso a errores.

Sin embargo, los servicios web nos ofrecen una mejora sobre la segunda posibilidad, al proporcionar un mecanismo estandarizado para intercambiar datos entre aplicaciones.

Los servicios web facilitan la tarea de construir programas que puedan intercambiarse datos incluso entre distintos lenguajes así como facilitar dicho intercambio entre aplicaciones que ya se hubiesen creado sin pensar en esta integración.

El hecho de que los servicios web utilicen HTTP para enviar y recibir mensajes implica que los mensajes del protocolo de nivel de aplicación de un servicio web están, a su vez, encapsulados dentro de peticiones y respuestas HTTP. Dicho de otro modo, cuando un cliente de un servicio web realiza una petición a un servidor, esta se envía en el interior de un mensaje HTTP, como un GET, POST, o similar. A su vez, debemos recordar que HTTP utiliza habitualmente TCP como protocolo de transporte, por lo que estos mensajes irán también dentro de un mensaje TCP.

Según Wikipedia:

Un **servicio web** (en inglés, *web service* o *web services*) es una tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web.

El **W3C** define un servicio web como:

> Un servicio web es un sistema software diseñado para soportar la interacción máquina-a-máquina, a través de una red, de forma interoperable.

7.1. Ventajas de los servicios web

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

La principal razón para usar servicios Web es que se pueden utilizar con HTTP sobre *Transmission Control Protocol* (TCP) en el puerto de red 80. Dado que las organizaciones protegen sus redes mediante *firewalls* (que filtran y bloquean gran parte del tráfico de Internet), cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores web. Los servicios Web utilizan este puerto, por la simple razón de que no resultan bloqueados. Es importante señalar que los servicios web se pueden utilizar sobre cualquier protocolo, sin embargo, TCP es el más común.

Una tercera razón por la que los servicios Web son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más utilizada.

8. Arquitectura orientada a servicios (SOA, siglas del inglés *Service Oriented Architecture*)

La orientación a servicios es una forma de pensar en servicios, su construcción y sus resultados. Un servicio es una representación lógica de una actividad de negocio que tiene un resultado de negocio específico (ejemplo: comprobar el crédito de un cliente, obtener datos de clima, consolidar reportes de perforación)

El estilo de arquitectura SOA se caracteriza por:

- Estar basado en el diseño de servicios que reflejan las actividades del negocio en el mundo real, estas actividades forman parte de los procesos de negocio de la compañía.
- Representar los servicios utilizando descripciones de negocio para asignarles un contexto de negocio.
- Tener requerimientos de infraestructura específicos y únicos para este tipo de arquitectura, en general se recomienda el uso de estándares abiertos para la interoperabilidad y transparencia en la ubicación de servicios.
- Estar implementada de acuerdo con las condiciones específicas de la arquitectura de TI en cada compañía.
- Requerir un gobierno fuerte sobre las representación e implementación de servicios.
- Requerir un conjunto de pruebas que determinen que es un buen servicio.

Existen diversos estándares relacionados con los servicios web; incluyendo los siguientes:

- **XML**
- **HTTP**
- **SOAP**
- **REST**
- **WSDL**
- **UDDI**

Nosotros tratamos sobre todo los servicios web SOAP y servicios web REST.

8.1. Servicios Web SOAP

El primer tipo de servicios web que existieron fueron los llamados “servicios web SOAP”. Reciben este nombre debido al formato en que representan los mensajes, que siguen el estándar SOAP (en inglés Simple Object Access Protocol). En SOAP se utiliza el lenguaje XML para definir tanto el protocolo de mensajes como el

contenido de estos. Esto hace que los mensajes sean fácilmente procesables de forma automática, a la vez que permite una alta flexibilidad a la hora de diseñarlos. La facilidad de procesamiento de los mensajes XML facilita además el desarrollo separado de servidores y clientes, y la fácil adaptación de las aplicaciones existentes, independientemente del lenguaje de programación en el que estuviesen desarrolladas. La descripción de interfaz de servicio de un servicio web SOAP se realiza usando un lenguaje basado en XML llamado WSDL (en inglés Web Services Description Languaje).
Un servicio web basado en SOAP debe cumplir los siguientes requisitos:

- Se debe expresar de manera formal y pública la interfaz del servicio. Esto se suele hacer usando WSDL.
- La arquitectura del servicio debe ser capaz de soportar realización y procesamiento de peticiones de forma asíncrona.
- Los servicios web basados en SOAP pueden o no tener estado (stateful o stateless).

8.2. Servicios Web REST

El término *REST* se usa en el sentido más amplio para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP.

REST afirma que la web ha disfrutado de escalabilidad como resultado de una serie de diseños fundamentales clave:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)

- Un conjunto de operaciones bien definidas que se aplican a todos los *recursos* de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (CLAB en castellano: crear,leer,actualizar,borrar) que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.

- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso es direccionable únicamente a través de su URI.

- El uso de hipermedios, tanto para la información de la aplicación como para las transiciones de estado de la aplicación: la representación de este estado en un sistema REST son típicamente HTML o XML. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

8.3. Diferencias con SOAP

Mientras que una arquitectura REST está fundamentalmente centrada en datos (recursos), una arquitectura que use el protocolo SOAP se orienta más hacia los servicios que permiten operar con dichos datos.

Los clientes REST envían información generalmente en HTML o XML, mientras que una implementación con SOAP se decanta normalmente por el último; teniendo un fuerte tipado, con un lenguaje más verboso que JSON –muy extendido en servicios que implementan arquitecturas REST– y demandando un mayor ancho de banda por transmisión.

Por otra parte, los servicios REST deben poder ser probados mediante un navegador para poder ser RESTful, por lo que el propio servicio debe controlar si devuelve un contenido solo amigable para las máquinas o una presentación también legible para humanos en función del contexto. Para probar una implementación de SOAP, se hace necesario el uso de herramientas *ad-hoc* como la conocida SoapUI. Además, el uso de bibliotecas está más extendido en esta última opción, quedando REST reservado únicamente para las URIs que representan los recursos como se ha descrito anteriormente.

REST se basa en el principio de lectura como operación más frecuente, por lo que la mayor parte de las peticiones serán de tipo GET; mientras que SOAP está más construido con peticiones POST. Debido a esto último y a unos recursos inespecíficos, SOAP puede conllevar el desarrollo de clientes más complejos que los que se podrían construir con servicios REST.