

Apuntes-SI1-Tema-01-y-02.pdf

**Anónimo**

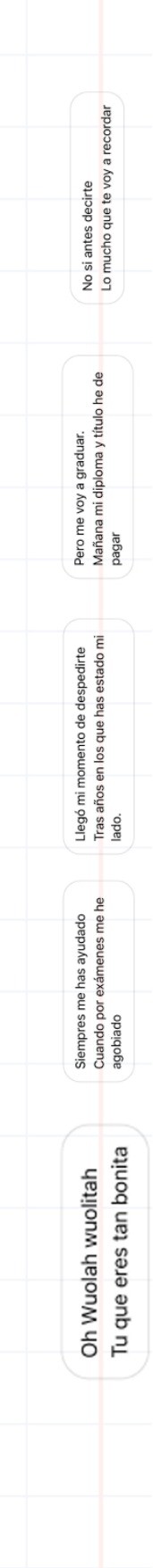
**Sistemas Informaticos I**

**3º Grado en Ingeniería Informática**

**Escuela Politécnica Superior Universidad Autónoma de Madrid**

Reservados todos los derechos.

**No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.**



**Tema 01 Introducción a los Sistemas Distribuidos**

* + - **1.1 DEFINICIONES Y CONCEPTOS BÁSICOS**

### SISTEMAS DISTRIBUIDOS. DEFINICIONES

Un sistema distribuido es una colección de elementos de computación autónomos (nodos) que cooperan entre sí para resolver un problema. Sistema complejo.

En el contexto de la asignatura:

**Sistema centralizado:**

* + - Ordenador central y red de terminales sin capacidad de proceso.

**Sistema distribuido:**

* + - Conjunto de elementos de proceso computacional independientes
    - no necesariamente homogéneos
    - están interconectados por una red de comunicaciones de cualquier tipo
    - cooperan mediante el envío de mensajes para realizar las tareas que tienen asignadas
    - dan la apariencia de sistema único y coherente a sus usuarios.

### MOTIVACIÓN DE LOS SISTEMAS DISTRIBUIDOS

* + - Distribución inherente de algunas aplicaciones
    - Compartición de recursos
    - Acceso a recursos remotos
    - Economía: mejora la relación rendimiento/coste
      * Ley de Grosh (~70s): la potencia computacional de una CPU es proporcional al cuadrado de su precio. A día de hoy, la ley de Grosh no es válida como tal, p.ej., computación en la nube.
    - Incremento de la potencia computacional, la capacidad de crecimiento y la velocidad de cálculo
    - Procesamiento paralelo
    - Escalabilidad
    - Flexibilidad y modularidad

### DESVENTAJAS DE LOS SISTEMAS DISTRIBUIDOS

* + - Aumenta la complejidad: cambian los modelos arquitectónicos
    - Gestión del tiempo
    - Coste computacional de las comunicaciones
    - Las comunicaciones son fuente de problemas/error:
    - Pérdida de mensajes
    - Saturación
    - Latencia
    - Seguridad
    - Confidencialidad



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

**¿QUÉ DISTRIBUIR EN UN SI?**

* Lógica de proceso
* Funciones
* Datos
* Control
* etc

## TRANSPARENCIA EN SISTEMAS DISTRIBUIDOS

* Da apariencia de sistema único y coherente a sus usuarios
* Acceso: el usuario accede a los recursos como si fueran locales
* Localización: no importa donde están los recursos, pudiendo moverse en caliente (relocalización) y en frío (migración) sin que esto afecta al sistema
* Replicación: se crean réplicas de los recursos de forma transparente al usuario
* Concurrencia: varios usuarios acceden simultáneamente sin entrar en conflicto
* Fallos: el sistema oculta y corrige los fallos
* Escalado: el sistema se expande o reduce (automática o manualmente) sin afectar a las aplicaciones
* La transparencia completa es difícil de conseguir y en la práctica muchas veces no es deseable.
  + Es complejo, y a veces imposible, ocultar los fallos de ciertas partes del sistema
  + La latencia no se puede ocultar

## COORDINACIÓN EN LOS SISTEMAS DISTRIBUIDOS

* Colección de nodos que cooperan entre sí intercambiando mensajes
* Coordinación hace referencia a cómo se comunican y coordinan los nodos:
  + Acoplamiento temporal
  + Acoplamiento referencial (espacial)
* Coordinación directa: acoplamiento referencial y temporal
* Distintos modelos arquitectónicos de sistemas distribuidos consiguen flexibilidad mediante modelos de coordinación indirecta:
  + Desacoplamiento referencial: los nodos no se conocen de forma explícita
  + Desacoplamiento temporal: los nodos no tienen que estar necesariamente activos de forma simultánea para coordinarse.

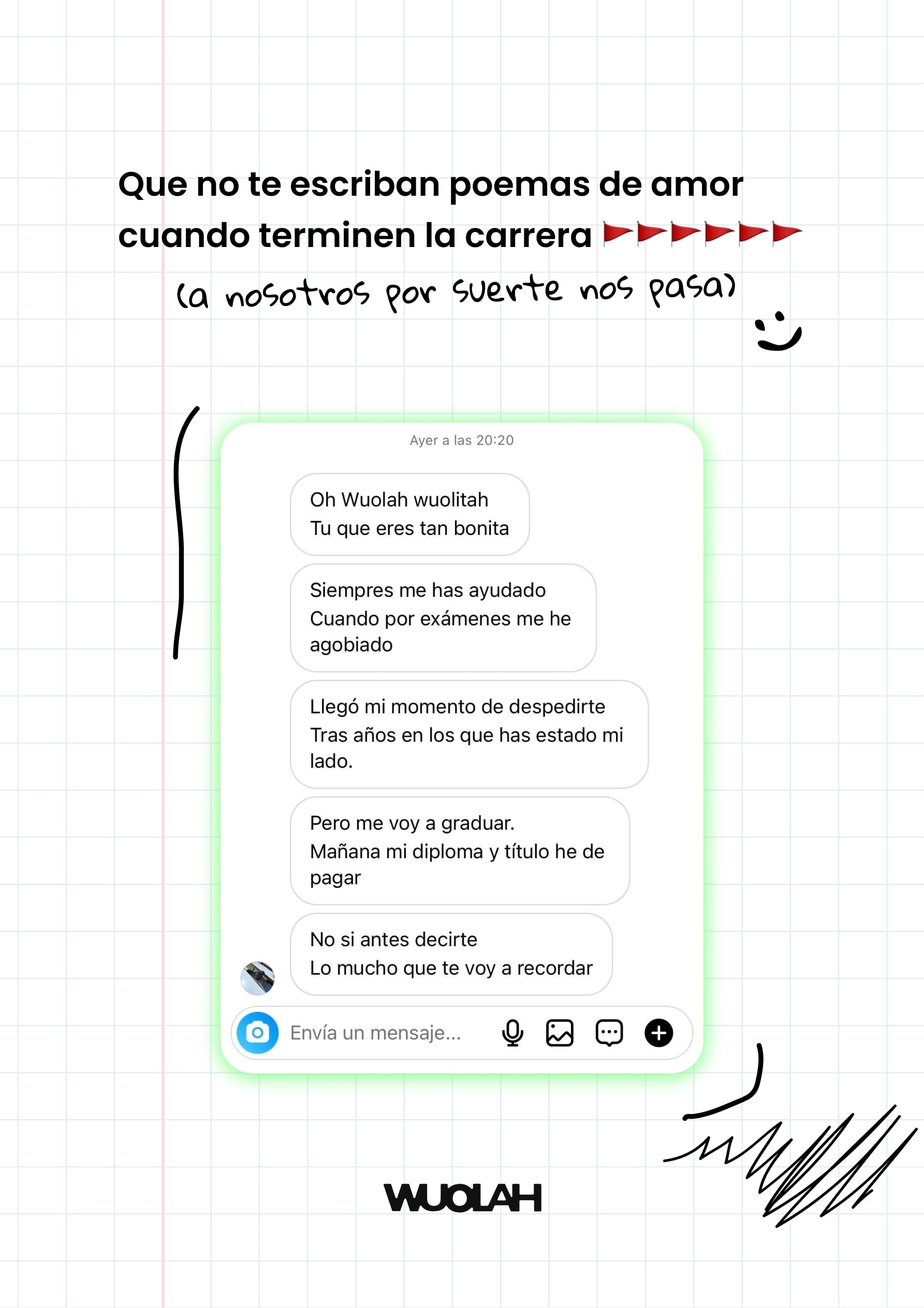
## TIPOS DE SISTEMAS DISTRIBUIDOS

Atendiendo a su grado de acoplamiento (HW):

* Fuertemente acoplados: Procesadores que comparten memoria o buses de entrada/salida. Aplicaciones multiprocesador
* Débilmente acoplados: Procesadores autónomos interconectados por sistemas de comunicaciones Atendiendo a su arquitectura software típicamente distinguimos dos tipos:
* Igual a igual (peer to peer, p2p):
  + Sistema simétrico
  + Todos los procesos desempeñan tareas semejantes



#### [si lees esto me debes un besito](https://wlh.es/v2/1685443802005/aHR0cHM6Ly9hcGkud3VvbGFoLmNvbS9wdWIvY2xpY2s_)



* + Interactúan para realizar una actividad distribuida
  + Interacción N-N
* Cliente-servidor:
  + Sistema asimétrico
  + Procesos clientes solicitan servicios
  + Procesos servidores los ejecutan y devuelven los resultados
  + Interacción N-1

# ARQUITECTURA DE LOS SISTEMAS DISTRIBUIDOS

## ARQUITECTURA DE LOS SISTEMAS DISTRIBUIDOS

La organización lógica de los componentes de un sistema distribuido varía con cada aplicación, pero existen patrones que se repiten habitualmente.

Típicamente se consideran dos arquitecturas SW básicas para los sistemas distribuidos:

* Igual a igual (peer to peer, p2p)
* Cliente-Servidor
  + Históricamente es la arquitectura más importante
  + Continúa siendo la más ampliamente utilizada (o alguna de sus variantes)

## ARQUITECTURA SOFTWARE

La arquitectura software de un SI define el sistema en términos de componentes computacionales e interacciones entre ellos

Identifica:

* Los componentes SW del sistema
* La ubicación de cada componente en la red
* Las interrelaciones entre componentes (conectores)
* La distribución de datos y tareas computacionales para configurar el SI

Evalúa el rendimiento, confiabilidad, escalabilidad y otras propiedades del sistema de software Componentes:

* clientes
* servidores
* bases de datos
* ...

Interacciones:

* llamadas a procedimientos (p.ej., RPC o RMI)
* mensajes
* compartición de variables
* protocolos cliente/servidor
* protocolos de acceso a BB.DD.
* streaming
* ...

## ¿POR QUÉ ES IMPORTANTE LA ARQUITECTURA SOFTWARE DE UN SI?

* Forma la columna vertebral para construir un sistema de software
* Es en gran medida responsable de permitir o no ciertos atributos de calidad del sistema (p.ej., confiabilidad y rendimiento)
* Es un modelo abstracto reutilizable
  + Puede transferirse de un sistema a otro
  + Representa una inversión
* Representa un medio de comunicación y discusión entre participantes del proyecto

**ARQUITECTURA DE LOS SISTEMAS DISTRIBUIDOS**

* La organización lógica de los componentes de un sistema distribuido varía con cada aplicación, pero existen patrones que se repiten habitualmente
* Típicamente se consideran dos arquitecturas SW básicas para los sistemas distribuidos:
  + Igual a igual (peer to peer, p2p)
  + Cliente-Servidor
    - Históricamente es la arquitectura más importante
    - Continúa siendo la más ampliamente utilizada (o alguna de sus variantes)
* Objetivos de diseño:
  + Compartir y optimizar el uso de recursos
  + Transparencia
  + Sistema abierto (openness)
  + Escalabilidad

## ARQUITECTURA DE LOS SD. CLAVES DEL DISEÑO

Objetivos de diseño:

* Compartir y optimizar el uso de recursos
* Transparencia
* Sistema abierto (openness)
* Escalabilidad

Durante el diseño es un error con consecuencias muy costosas de solucionar asumir que:

* Las comunicaciones son confiables en todo momento y en cualquier circunstancia
* Las comunicaciones son seguras
* No hay restricciones de ancho de banda
* La red/sistema de comunicación es homogéneo
* La infraestructura/topología no cambia
* La latencia es despreciable
* El coste añadido por la capa de transporte es despreciable
* El coste económico de una infraestructura distribuida es siempre menor
* El administrador es único

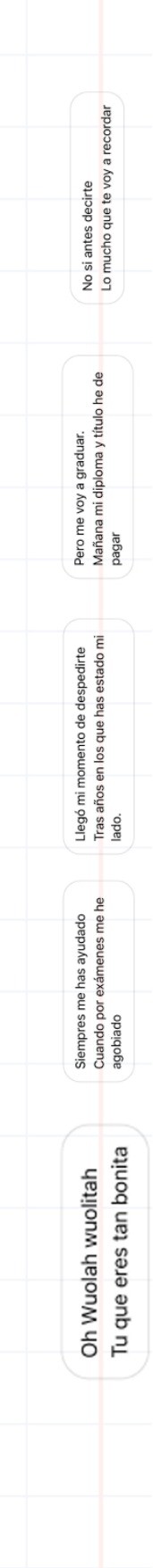
## MODELOS ARQUITECTÓNICOS: CAPAS VS. NIVELES

Diagrama

Descripción generada automáticamenteCapa: se refiere a la arquitectura SW del sistema (capa lógica, layer) Nivel: se refiere a la arquitectura HW del sistema (nivel físico, tier)

(Imagen de arquitectura de capas) Ej arquitectura capas: protocolo de

comunicación (capa transporte, red, etc).



## ARQUITECTURA TRADICIONAL EN TRES CAPAS

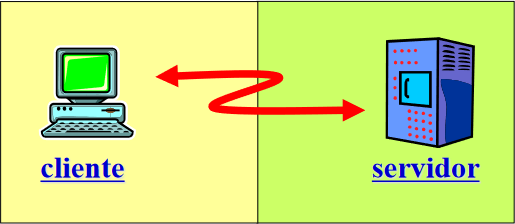
El sistemas informático consta de tres capas funcionales desacopladas entre las que se reparten los distintos componentes de la aplicación:

* Capa de interfaz de usuario-aplicación (normalmente, gráfica)
* Capa de procesamiento: lógica de aplicación
* Capa de datos: datos que el cliente manipula a través de la aplicación

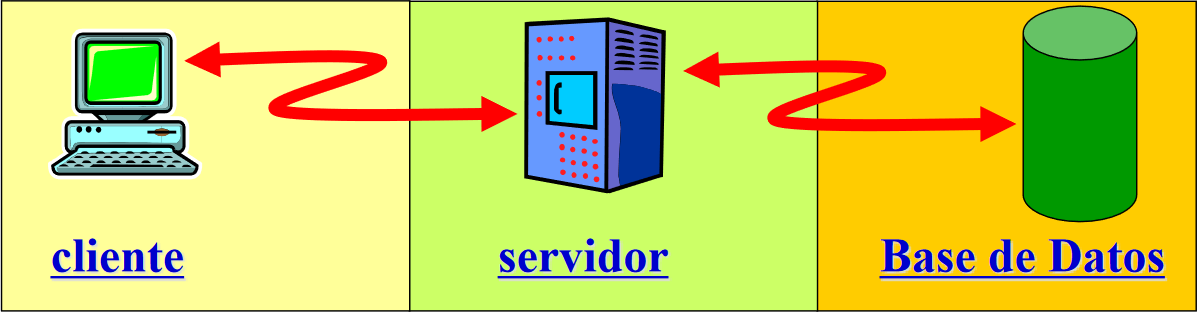
Esta arquitectura básica se puede generalizar a más capas, conforme la aplicación se estructura con un modelo de componentes funcionales desacoplados

* Por ejemplo, podemos hablar de una arquitectura en cuatro capas si entre la capa de procesamiento y la de datos añadimos una capa de acceso a datos

### APLICACIONES CLIENTE-SERVIDOR DE 2 NIVELES (2-TIER)

* La lógica de la aplicación se ejecuta junto con la lógica de presentación (modelo cliente pesado)
* El servidor realiza acceso a datos
* Implementación sencilla para aplicaciones pequeñas
* Ampliación de la aplicación y migración a otros entornos compleja

### APLICACIONES CLIENTE-SERVIDOR DE 3 NIVELES (3-TIER)

* Lógica de aplicación y de presentación separadas
* La lógica de la aplicación reside en cualquiera de los elementos de la red
* Aplicaciones más robustas y fácilmente escalables

### GENERALIZACIÓN: APLICACIONES EN N NIVELES (N-TIER)

* Múltiples niveles de clientes y servidores.
* Representan nuevos modelos de arquitecturas de sistemas distribuidos basados en la arquitectura cliente-servidor.

javas



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

## ARQUITECTURAS ORIENTADAS A OBJETOS Y SERVICIOS

El sistema distribuido se compone de objetos (≡ componentes) que se conectan mediante llamadas a procedimientos

* Los objetos pueden estar en máquinas diferentes llamadas a procedimientos remotos
* Los objetos encapsulan datos y ofrecen métodos para manipularlos sin revelar la implementación interna objetos con estado y objetos remotos.
* SOA: es la evolución de la arquitectura basada en objetos hacia el negocio

## ARQUITECTURAS ORIENTADAS A RECURSOS

Surgen para facilitar la integración de servicios en arquitecturas SOA

* El sistema distribuido se ve como una colección de recursos, gestionados individualmente por componentes
* Los recursos puede ser añadidos, eliminados, obtenidos y modificados remota y dinámicamente por las aplicaciones
* Ampliamente utilizado en la actualidad en los sistemas web arquitecturas RESTful (Representational State Transfer):
* Los recursos se identifican de forma única (URI)
* Todos los servicios ofrecen la misma interfaz basada en mensajes HTTP:
  + POST: Crea un recurso
  + GET: Obtiene una representación del recurso (o su estado)
  + DELETE: Borra un recurso
  + PUT: Modifica un recurso
* Los mensajes enviados a y desde los servicios tienen la descripción de la operación
* Los servicios no tienen memoria (stateless)

## ARQUITECTURAS BASADAS EN EVENTOS

Se trata de separar el procesamiento y la coordinación. Otras arquitecturas consideran la coordinación directa.

El sistema es una colección de procesos que operan de manera autónoma:

* Desacoplamiento temporal: Sistemas asíncronos, donde el procesamiento (respuesta) puede darse un espacio de tiempo variable después de la llamada
* Desacoplamiento referencial: No es necesario conocer de antemano la referencia
* Concepto de publicación y suscripción

## COMPONENTES SW DE UN SISTEMA DISTRIBUIDO

En los sistemas distribuidos el SW suele ser más decisivo que el HW

## COMPONENTES SW DE UN SISTEMA CLIENTE-SERVIDOR

Se suelen considerar tres módulos principales:

* Cliente. Solicita/consume servicios
* Middleware. Enlace entre cliente y servidor o entre servidores. Fundamental para construir sistemas abierto
* Servidor. Proporciona servicios

## CLIENTES

* Cliente es todo proceso que reclama/consume servicios ofrecidos por otros.
* No es equivalente a aplicación que interacciona con el usuario.
* Un servidor, a su vez, puede ser cliente de otros servidores.
* La funcionalidad del proceso cliente marca la operativa de la aplicación (flujo de información o lógica de negocio)

## SERVIDORES

Servidor es todo proceso que proporciona un servicio a otros

* Servidores de disco
* Servidores de impresión
* Servidores de comunicaciones
* Servidores de presentación
* Servidores de procesos
* Servidores de bases de datos
* Servidores de seguridad
* etc

Ambigüedad: Normalmente se llama “servidor” tanto a las aplicaciones o procesos que realizan un servicio como a los ordenadores que les sirven de soporte. En esta asignatura normalmente nos referiremos a los procesos de servicio.

## MIDDLEWARE

* Puede verse como el “sistema operativo” de los sistemas distribuidos
* Se ejecuta sobre el sistema operativo local tanto en el servidor como en el cliente
* No depende ni del HW ni del S.O. subyacente
* Interfaz Cliente/Servidor o Servidor/Servidor

## MIDDLEWARE: PATRONES DE DISEÑO

Tres capas:

* Protocolo específico del servicio
* Network Operating System, NOS
* Protocolo de transporte Adaptadores (wrappers) vs. interceptores

## MODELOS DE SISTEMAS DISTRIBUIDOS

* Red de compartición de recursos
* Servidores de base de datos
* Proceso de transacciones
* Sistemas de soporte de trabajo en grupo
* Sistemas de objetos distribuidos
* Servicios Web
* Sistemas distribuidos con clientes basados en la WWW

## SERVIDORES DE BASE DE DATOS

* El cliente pasa una petición (query) SQL en un mensajes al servidor (Data Base Management Server, DBMS)
* Los resultados de cada consulta SQL se devuelven por la red
* El código que ejecuta la petición SQL se encuentra en el mismo ordenador que los datos

## PROCESO DE TRANSACCIONES

1. El cliente solicita la ejecución de un procedimiento remoto en el servidor
2. El procedimiento ejecuta un grupo de peticiones SQL o de otro tipo (transacción)
3. Todas las peticiones se ejecutan o fallan como una unidad
4. El resultado final se devuelve al cliente
5. La aplicación se desarrolla escribiendo el cliente y el código de las transacciones en el servidor
6. Online Transaction Processing, OLTP

## SERVICIOS WEB

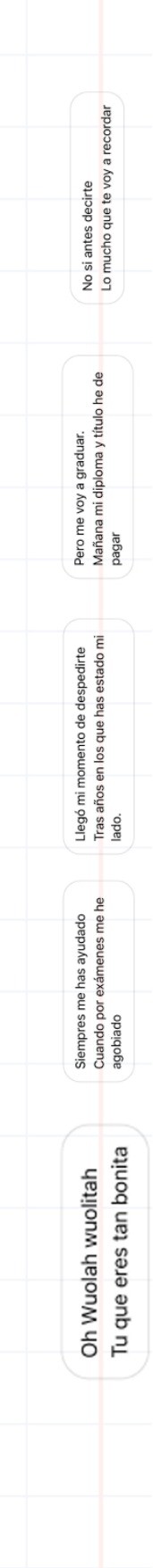
* Middleware con modelo de llamadas a procedimientos remotos sobre WWW
  + Publicación de servicios por parte de los servidores
  + Descubrimiento dinámico de servicios por los clientes
  + Ejecución de los servicios por los clientes siguiendo un modelo RPC
* Modelos de ejecución punto a punto o pasando a través de un intermediario (broker) único: Enterprise Service Bus
* Base para la Services Oriented Architecture (SOA)

# CARACTERÍSTICAS DE LOS SISTEMAS CLIENTE-SERVIDOR

## ARQUITECTURA CLIENTE-SERVIDOR

Objetivo principal: proveer una arquitectura escalable Relación asimétrica en cada interacción entre componentes Servidor:

* En el modelo básico, típicamente pasivo
* Espera peticiones
* Cuando recibe una petición, la procesa y envía respuesta
* Puede conservar o no el estado de la comunicación Cliente:
* En el modelo básico, típicamente activo
* Envía peticiones
* Espera hasta que llega la respuesta



### ARQUITECTURA CLIENTE-SERVIDOR CARACTERÍSTICAS BÁSICAS

* Sistemas débilmente acoplados. Interacción basada en el envío de mensajes
* Servicio. Unidad básica de diseño. El servidor los proporciona y el cliente los utiliza
* Encapsulamiento de servicios. Los detalles de la implementación de un servicio son transparentes para el cliente
  + Lleva a las arquitecturas basadas en servicios (SOA)
* Recursos compartidos. Muchos clientes utilizan los mismos servidores y, a través de ellos, comparten recursos lógicos o físicos
* Protocolos de aplicación asimétricos. En su esquema básico, relación 1-n
  + El servidor es un cuello de botella: activo-pasivo y/o escalado horizontal
* Transparencia. El sistema aparece como una única unidad de proceso. Independencia de la plataforma HW y SW que se emplee
* Escalabilidad, horizontal (requiere balanceo de carga) y vertical
* Datos y programas centralizados en servidores facilitan su integridad y mantenimiento

### WWW: EJEMPLO PARADIGMÁTICO DE ARQUITECTURA CLIENTE-SERVIDOR

En el modelo arquitectónico WWW tradicional:

* El cliente realiza una petición utilizando el protocolo HTTP o HTTPS
* El servidor responde con un documento codificado en HTML

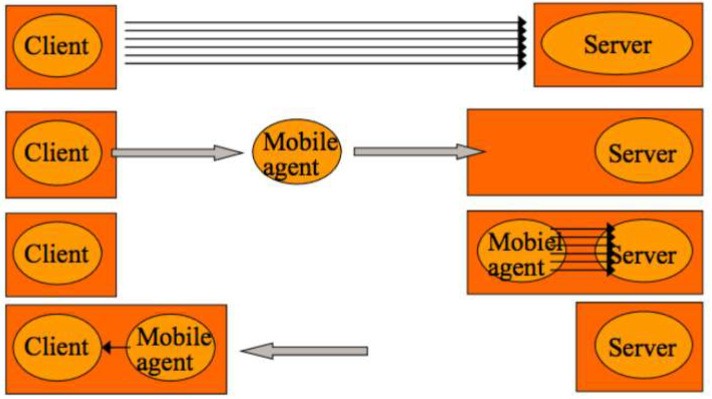
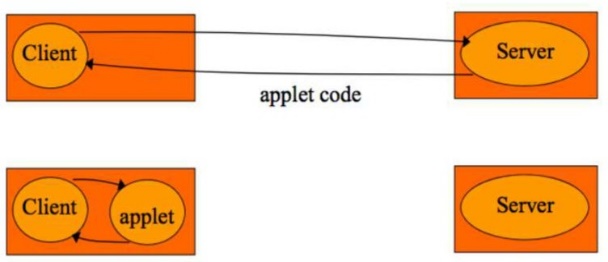
### VARIANTES DEL MODELO CLIENTE-SERVIDOR

Servicio proporcionado por múltiples servidores y servidores jerárquicos

|  |  |
| --- | --- |
| Servicio proporcionado por múltiples servidores y servidores jerárquicos | Servidores proxy y cachés |
|  |  |
| Código móvil | Agentes móviles |



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.



# ARQUITECTURA DE LAS APLICACIONES WWW

## SISTEMAS DISTRIBUIDOS BASADOS EN LA WORLD WIDE WEB

En origen, sistemas distribuidos basados en la extensión del modelo de cliente ligero bajo protocolo HTTP para el intercambio de información

Clientes:

* Universales: navegadores web (Web Browsers)
* Específicos: programas ejecutados en el cliente Servidores Web + Servidores de Aplicaciones:
* Repositorios de documentos
* Entorno de ejecución de aplicaciones Servidores de Back-End:
* Enlace con programas ya existentes o desarrollos antiguos (legacy systems)
* Programas en el servidor Web les realizan consultas a través de algún tipo de middleware más o menos elaborado
* Los casos más comunes son:
  + Servidores de bases de datos
  + Servidores de proceso de transacciones.

## WORLD WIDE WEB VS. INTERNET

Aunque en contextos no técnicos muchas veces se usan de forma equivalente, hacen referencia a conceptos muy distintos

Diagrama

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

Ejemplo aplicación web

Arquitectura básica

## 1.5 INTRODUCCIÓN A LA COMPUTACIÓN EN LA NUBE

## INTRODUCCIÓN A LA COMPUTACIÓN EN LA NUBE

La computación en la nube (cloud computing) es una solución tecnológica que en los últimos años ha ganado en popularidad y se ha convertido en tendencia para el desarrollo y despliegue de sistemas distribuidos

Pero, ¿qué es la computación en la nube?

* Almacenamiento y acceso a través de internet a datos y programas en una ubicación remota (≠ máquina remota)
* Modelo que permite el acceso ubicuo, simple y bajo demanda a un grupo compartido y configurable de recursos computacionales (servidores, almacenamiento, aplicaciones o servicios) que pueden ser provisionados y desplegados con un esfuerzo de administración mínimo o con la mínima intervención del proveedor que da dichos servicios [National Institute of Standards and Technologies] 5-4-3 principios de la computación en la nube
* Definición e implementación de sistemas distribuidos en los que todos los recursos HW y SW son ofrecidos como un servicio (normalmente de pago) por un tercero (cloud service provider)

Un dibujo animado con letras

Descripción generada automáticamente con confianza media

EJEMPLOS DE CLOUD SERVICE PROVIDERS

## VENTAJAS Y DESVENTAJAS DE LA COMPUTACIÓN EN LA NUBE

Ventajas

* Disminución de costes
  + Una infraestructura 100% en la nube no requiere “instalar” ningún tipo de HW/SW
  + pay-as-you-go, pay-as-per-use
* Permite desarrollos complejos sin necesidad de un conocimiento avanzado
  + En muchas ocasiones el uso de un servicio simplemente requiere de su customización
  + Rapidez de desarrollo y con menos riesgo
  + Acceso a nuevas tecnologías
* Administración
  + Tiempo y recursos/coste
  + Alta disponibilidad

Res

* + Actualización de sistemas
  + Escalabilidad

Desventajas

* Privacidad
* Falta de control y dependencia del proveedor

## COMPUTACIÓN EN LA NUBE. 5 CARACTERÍSTICAS ESENCIALES

1. Servicios bajo demanda. Las capacidades (procesamiento y recursos físicos o virtuales) se adquieren bajo demanda en función de las necesidades
2. Amplio acceso de red. Los servicios se encuentran disponibles en una red que puede ser privada, compartida o pública, siendo accesibles a través de mecanismos estándar que permiten su uso por clientes ligeros heterogéneos
3. Pooling de recursos. Las capacidades se asignan y reasignan dinámicamente entre los distintos “clientes” en función de la demanda
4. Rápida elasticidad. El escalado horizontal y vertical de capacidades debe producirse rápidamente (en algunos casos incluso de forma automática)
5. Medición de servicios. El uso de recursos debe monitorizarse y reportarse de manera automática de cara a su control y optimización

## COMPUTACIÓN EN LA NUBE. 4 MODELOS DE DESPLIEGUE

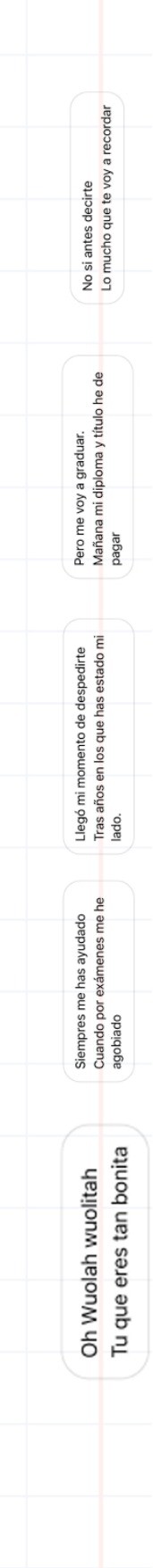
1. Nube privada. Sólo una organización tiene acceso a la infraestructura
2. Nube pública. El acceso a la infraestructura es abierto
3. Nube comunitaria. La infraestructura es compartida por varias organizaciones
4. Nube híbrida. La infraestructura general se compone de dos o más infraestructuras con distintos modelos de despliegue

## COMPUTACIÓN EN LA NUBE. 3 MODELOS DE SERVICIO

1. Infraestructura (IaaS): Se proporciona capacidad de procesamiento, almacenamiento, red y otros recursos computacionales fundamentales (servidores, sistemas operativos, virtualización...). Esto permite desplegar y ejecutar cualquier software arbitrario. El proveedor del servicio es el dueño del equipamiento y el responsable del housing y el mantenimiento. El “cliente” controla y administra el sistema operativo, las aplicaciones, los datos, y ciertos componentes de la red (p.ej., firewalls)
2. Plataforma (PaaS): Este tipo de arquitectura está orientada principalmente a desarrolladores. Ofrece un entorno preconfigurado de desarrollo/ejecución usando los lenguajes de programación, librerías, servicios y herramientas soportados por el proveedor. El “cliente” no gestiona, ni controla la infraestructura
3. Software (SaaS): Se da acceso a las aplicaciones que el proveedor ejecuta en su infraestructura, sin tener ningún control sobre ésta

Diagrama

Descripción generada automáticamente



**Tema 02 SISTEMAS DISTRIBUIDOS BASADOS EN LA WORLD WIDE WEB**

* 1. **INTRODUCCIÓN**

### SISTEMAS DISTRIBUIDOS BASADOS EN LA WORLD WIDE WEB

En origen, sistemas distribuidos basados en la extensión del modelo de cliente ligero bajo protocolo HTTP para el intercambio de información

La generalización del uso de la WWW como mecanismo de intercambio de información produjo un cambio en el modo de diseñar las aplicaciones distribuidas

* + - El cliente Web se convierte en la interfaz de uso genérico para la interacción de los usuarios con un sistema distribuido
      * En el modelo actual, las carencias del modelo de cliente ligero universal se suplen con elementos de programación complementarios
    - El servidor Web pasa a ser el núcleo básico donde se gestiona la ejecución de programas. Es necesario completar su funcionalidad inicial para:
      * Ejecutar programas (~ servidor de Aplicaciones)
      * Enlazar con programas ya existentes (servidor de Back-end)

En este tema se tratan los distintos modos que se han ido desarrollando para implementar este modelo de sistemas hasta llegar al modelo actual

### INTERNET, INTRANET, EXTRANET

Los tres son términos que se refieren a redes de comunicación físicas que principalmente se diferencian por el tipo de acceso, los usuarios que tienen acceso a ella y el tipo de información que almacena:

* + - Internet: Red pública, externa, comunicación con el mercado. Soporte business to customer (b2c)
    - Intranet: Red IP privada. Soporte de comunicaciones de un grupo de usuarios: empresa, institución...
      * Soporte business to employee (b2e)
    - Extranet: Acceso a una Intranet por usuarios externos: Clientes, proveedores…
      * Soporte business to business (b2b)

A nivel de arquitectura básica, no existen diferencias entre los sistemas distribuidos diseñados para internet, una intranet o una extranet:

* + - Sus diferencias provienen fundamentalmente de los distintas necesidades de seguridad.

### ORIGEN WORLD WIDE WEB

Tim Berners-Lee: Padre de la World Wide Web

* + - HTML
    - HTTP
    - URL (URI)
    - Primera comunicación entre un cliente y un servidor HTTP (1989)



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

* Fundador de la W3C

**WORLD WIDE WEB**

* Colección de documentos
* Cada documento puede contener objetos de diversos tipos:
  + Texto, gráficos, imágenes, voz, vídeo, otros documentos...
* Los objetos se reconocen mediante su Uniform Resource Identifier, URI
* Cada documento puede tener referencias a cualquier otro objeto en la red. Se denominan hiperenlaces
* El cliente accede a los documentos a través de un visualizador (browser) o navegador cliente ligero
* Los nodos de la red presentan los documentos a través de un servidor Web
* Comunicación mediante protocolo HTTP (HyperText Transfer Protocol) y cada vez más HTTPS
* Contenido de los documentos descritos en lenguaje HTML (HyperText Markup Language). Actualmente HTML5

## URI: UNIFORM RESOURCE IDENTIFIER

URI, Uniform Resource Identifier (RFC 2396)

* URN (Uniform Resource Name): identifica un recurso o espacio de nombres (namespace). Ejemplo: urn:isbn:n-nn-nnnnnn-n
* URL (Uniform Resource Locator): indica dónde/cómo encontrar el recurso Extensión del nombre de un fichero que permite identificar recursos en Internet
* esquema:parte-dependiente-del-esquema
* La parte dependiente del esquema se basa en namespaces y subnamespaces En el caso de las URLs, contienen datos sobre:
* El mecanismo de acceso primario (protocolo cómo se transmite la información)
* La dirección del componente de la red que contiene el recurso
* El identificador del recurso dentro del componente que lo contiene Dos tipos de URLs:
* Absolutas
  + <http://www.webcon.com/~tbrown/computer.html>
* Relativas
  + img/br2.gif

## ESTRUCTURA DE UNA URL

Esquema: asociado con alguno de los protocolos de comunicación o servicios disponibles en Internet: ftp, http, gopher, mailto, news, file, telnet...

Parte-dependiente-del-esquema: datos necesarios para localizar el objeto

* En su mayoría, su namespace responde a la estructura:
  + //[usuario[:clave]@]host[:puerto]/[path]
* El parámetro puerto se puede omitir y se tomaría el puerto por defecto para el protocolo utilizado Ejemplos de namespaces URL:
* ftp://[usuario[:contraseña]@]host[:puerto]/[path-del-archivo]
  + ftp://ftp.ibm.com/programs/download/testip.zip
* http://[host][:puerto]/[path-recurso][?consulta]
  + [http://search.yahoo.com/bin/search?p=url+uri](http://search.yahoo.com/bin/search?p=url%2Buri)
* mailto:[Dirección-RFC822]
  + <mailto:bobsponja@fondobikini.com>

# WEB HIPERTEXTO

## WEB HIPERTEXTO

* Es el modelo original y más sencillo de la Web
* Documentos distribuidos en servidores en Internet
* Identificados por una URI
* Pueden contener otros objetos o documentos incrustados
* Enlaces desde cualquier documento a cualquier otro en la red
* El documento se define atendiendo a los elementos que contiene y no a la forma de presentarlo
  + Tendencia actual hacia una web semántica metadatos y ontologías
  + Qué vs. Cómo
* Clientes estándar para visualizar documentos

## HTTP: HYPERTEXT TRANSFER PROTOCOL

Protocolo de comunicaciones entre cliente y servidor Web

Transporte: Conexión TCP sobre el puerto 80 (por defecto, 443 HTTPS) Intercambio de mensajes ASCII entre ambos:

Cliente realiza una petición

GET /hypertext/www/TheProject.html HTTP/1.0

El servidor responde con un mensaje MIME (Multipurpose Internet Mail Extensions):

HTTP:/1.0 200 Document follows MIME-Version: 1.0 Server:CERN/3.0

Content-Type: text/html Content-Length: 8247

<HEAD><TITLE>The World Wide Web Consortium (W3C)</TITLE></HEAD>

<BODY>

<H1><IMB ALIGN=MIDDLE ALT="W3C" SRC="icons/WWW(w3c\_96x67.gif>

The World Wide Web Consortium</H1><P>

...

## PETICIONES HTTP

* Formato de las peticiones: [método] URI [protocolo]
* URI: Identifica el objeto sobre el que aplicar el método
* Protocolo: El del mensaje enviado: HTTP/1.0, HTTP/1.1
* Método: Acción a realizar:
  + GET: Lectura del objeto
  + HEAD: Lectura de la cabecera del objeto
  + PUT: Almacenamiento del objeto
  + POST: Añadir al objeto la información enviada
  + DELETE: Eliminar el objeto
  + LINK: Conectar el objeto con otro determinado
  + UNLINK: Desconectar el objeto de otro determinado

## FAMILIA DE LENGUAJES DE LA WWW

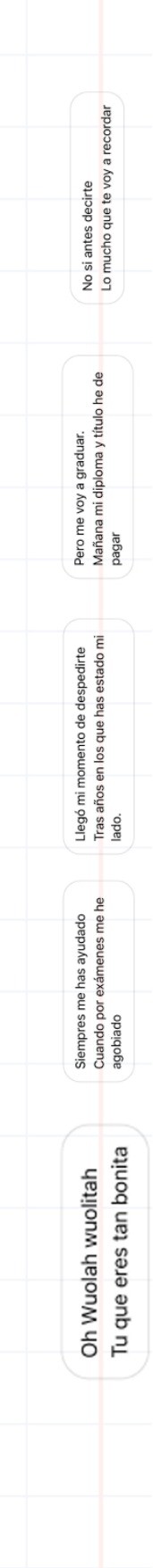
**SGML: STANDARD GENERALIZED MARKUP LANGUAGE**

Estándar ISO-8879-1/1986 para la definición de texto electrónico independiente de dispositivos, sistemas y aplicaciones

* Proviene de GML (IBM, 70’s)
* Metalenguaje para definir lenguajes de diseño descriptivos
* Proporciona un medio de codificar mensajes cuyo destino sea el intercambio directo entre sistemas o aplicaciones
* Múltiples lenguajes definidos mediante SGML
* Hypertext Markup Language, HTML
* Evolución de SGML: Extended Markup Language, XML

###### Características

* SGML tiene cinco características principales:
* Permite crear lenguajes de codificación descriptivos
* Códigos incluidos en el propio texto definen los elementos que lo componen
* Define una estructura de documento jerárquica, con elementos y componentes interconectados



* Proporciona una especificación formal completa del documento
* Contenida en la Document Type Definition, DTD
* No tiene un conjunto implícito de convenciones de señalización. Soporta, por tanto, un conjunto flexible de juegos de etiquetas
* SGML es un metalenguaje. Definición de etiquetas para cada lenguaje en su DTD
* Los documentos generados con él son legibles por personas

<!-- Parameter Entities -->

<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK" -- repeatable head elements -->

<!ENTITY % heading "H1|H2|H3|H4|H5|H6">

<!ENTITY % list "UL | OL | DIR | MENU">

<!ENTITY % preformatted "PRE">

....

<!ELEMENT FONT - - (%inline)\* -- local change to font -->

<!ATTLIST FONT

size CDATA #IMPLIED -- [+]nn e.g. size="+1", size=4 --

color CDATA #IMPLIED -- #RRGGBB in hex, e.g. red: "#FF0000" --

face CDATA #IMPLIED -- comma separated list of font names --

EJEMPLO SGML – DTD DE HTML 4

**HTML: HYPERTEXT MARKUP LANGUAGE**

Aplicación del estándar SGML. https://[www.w3.org/MarkUp/](http://www.w3.org/MarkUp/) Lenguaje de definición de formato de documentos

* Define los elementos que pueden aparecer dentro del documento
* Se especifican mediante etiquetas (Tags) de comienzo y final de documento
  + Texto ASCII encerrado entre < y >
* Sólo se define el tipo de elemento y no la forma de representarlo

Lenguaje estándar utilizado en la Web para el intercambio de documentos hipermedia que incluyen:

* Texto
* Imágenes (Fotos, Vídeo)
* Audio
* Vínculos (Links)

Estándar especificado por el World Wide Web Consortium, W3C

### ESTRUCTURA DE UN DOCUMENTO HTML

* Identificación SGML
* Permite identificar la DTD adecuada para procesarlo

##### <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">

* Delimitación del documento HTML
  + Identificado con la etiqueta <HTML>
* Cabecera. Información asociada al documento
  + Identificado por la etiqueta <HEAD>



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

* + Contiene información descriptiva del documento y definiciones de formato para los elementos del resto del mismo
* Cuerpo del documento
  + Identificado con la etiqueta <BODY>
  + Contiene los elementos de presentación del documento

Diagrama, Escala de tiempo

Descripción generada automáticamente

## COMPONENTES DEL LENGUAJE HTML

###### Etiquetas/Elementos

* No vacías:
  + Diagrama

    Descripción generada automáticamenteLa mayoría de elementos comienzan por un tag, luego contenido, finalizando por un tag de terminación

<TAG> contenido </TAG>

* Vacías: por ejemplo <LI>, <BR>, etc.

###### Atributos:

* Específicos de cada etiqueta
* Texto ASCII entre comillas (doble o simple)

dentro de la etiqueta de comienzo

* Pares clave-valor dentro de unos valores

permitidos HTML es insensible a Mayúsculas/Minúsculas:

<TAG> es equivalente a <tag> o <TaG>

Espacios en blanco, tabulaciones y retornos de carros no

son significativos

## SEPARACIÓN DE RESPONSABILIDADES

Una regla fundamental del diseño de software es que funciones distintas deben ser llevadas a cabo por entidades distintas

Separation of concerns

HTML originalmente fue diseñado para definir contenido de documentos

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

Pero no para definir formato, la forma en que se presentan los contenidos

Cuando se agregaron etiquetas como <font> (a partir de HTML 3.2) la cosa empezó a complicarse Se mezcla el contenido (el qué) con el formato (el cómo)!!!

## CSS: CASCADE STYLE SHEETS

Usado para describir la semántica de presentación de un documento escrito con un lenguaje de marcado El uso más común es darle estilo a páginas escritas en HTML y XHTML

El objetivo es separar el contenido del documento (HTML o similar) de cómo se presenta (layout, colores, fuentes, etc.)

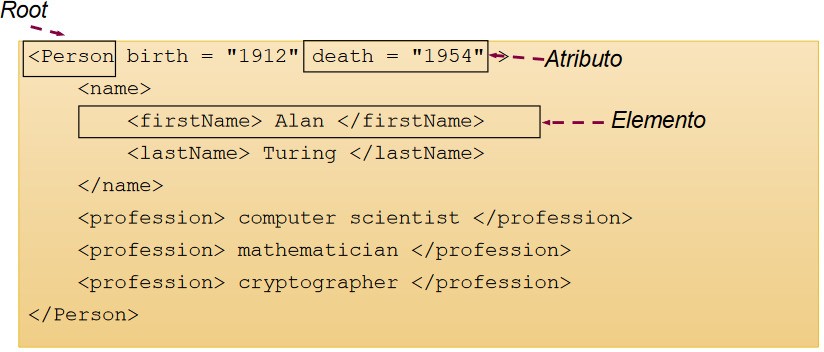
Ventajas:

* Forma rápida y sencilla de cambiar el aspecto del documento
* Ofrecer accesibilidad
* Más control y flexibilidad de presentación
* Compartir formato entre múltiples páginas

Res

* Distinta presentación para distinto tipo de dispositivo (ordenador, móvil, impreso...) o entre distinto tipo de usuarios de a aplicación (p.ej., usuarios de distintas compañías)

## XML: EXTENSIBLE MARKUP LANGUAGE

* Especificación del W3C ([www.w3.org/XML)](http://www.w3.org/XML)) para crear lenguajes descriptivos de propósito específico
  + Fecha de primera especificación: 1998
  + https://[www.w3.org/TR/1998/REC-xml-19980210](http://www.w3.org/TR/1998/REC-xml-19980210)
  + https://[www.w3schools.com/xml/xml\_whatis.asp](http://www.w3schools.com/xml/xml_whatis.asp)
  + XML is a software- and hardware-independent tool for storing and transporting data
  + XML Does Not DO Anything
* Procede de SGML reduciendo su complejidad
* Se puede ver como una generalización de HTML para modelar estructuras de datos de cara al procesamiento automático de información
* Lenguaje de meta-marcado
* Diseñado para describir estructuras de datos
* Utilizado para realizar intercambios de datos entre sistemas
* Etiquetas, elementos y atributos con estructura de árbol
* En los últimos años hay una tendencia a sustituir XML por JSON

## SINTAXIS XML

* El documento XML es de texto, normalmente Unicode
* Cada documento XML debe tener un único elemento raíz
* Todo elemento no vacío debe tener etiqueta de comienzo (<elemento>) y final (</elemento>)
* Todo elemento que no lleve contenido puede ser marcado con una etiqueta de elemento vacío (<elemento/>)
* Todos los valores de atributos deben ir entre comillas dobles
* Los elementos se pueden anidar, pero no solapar con otros
* Los nombres de los elementos son sensibles a mayúsculas y minúsculas
* Permite el uso de distintos espacios de nombres tomados de distintos vocabularios en el mismo documento (namespace:nombre)

## VALIDACIÓN CON DTD

Res

Document Type Definition

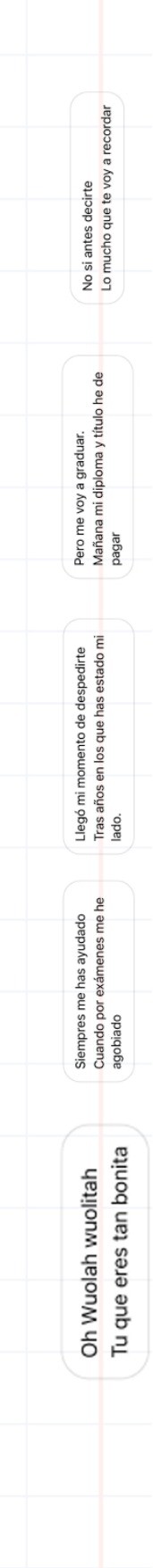
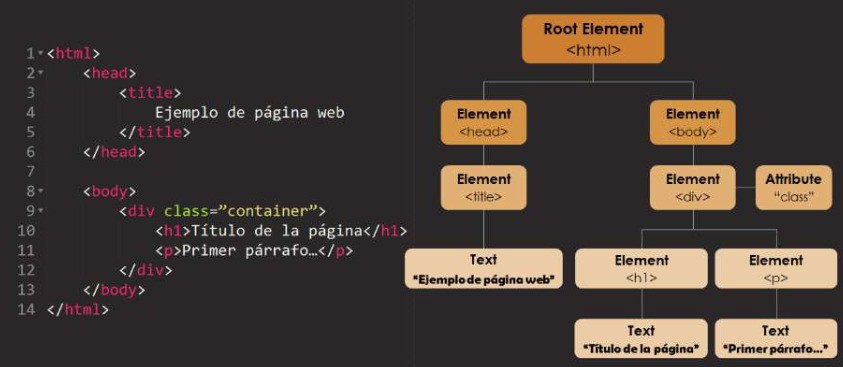
* No es obligatorio
* Especifican un sublenguaje de XML
  + Lista de etiquetas permitidas
  + Etiquetas y atributos permitidos dentro de cada etiqueta
* Pueden ser muy complejos
  + Para SVG (Scalable Vector Graphics) más de 1000 líneas
  + Para XHTML 1.0 más de 1500 líneas
  + Para DocBook más de 11000 líneas
* Alternativa: XML Schema (versión 1.0, mayo 2001)

## PARSEO DE DOCUMENTOS XML

SAX: Simple API for XML

* Orientado a eventos
* El programador proporciona métodos callback que son invocados por el parser a medida que lee el documento XML
* Es rápido y eficiente, pero difícil de usar
  + Cada nodo es tratado independientemente de lo que había antes o después
  + No se puede “volver atrás” en el procesado DOM: Document Object Model
* Los documentos se estructuran como árboles
* También sirve para otros lenguajes como HTML
* Estándar del W3C
* XML DOM:
  + Modelo de objetos estándar para XML
  + Una API estándar para XML, aunque no todos los parser DOM siguen el estándar
  + Independiente de la plataforma y del lenguaje de programación

Árbol DOM



### API DOM

Todo en DOM son nodos:

* + - El documento completo es un document node
    - Cada elemento XML es un element node
    - El texto en elementos XML son text nodes
    - Cada atributo es un attribute node
    - Los comentarios son comment nodes

La API DOM define una serie de propiedades y métodos que debe ofrecer cualquier implementación de DOM para acceder y manipular nodos en las distintas dimensiones del árbol DOM

* + - x.nodeName: el nombre de x
    - x.nodeValue: el valor de x
    - x.parentNode: el nodo padre de x
    - x.childNodes: los nodos hijos de x
    - x.attributes: los nodos atributos de x

### XPATH

XPATH es un mecanismo para definir expresiones para navegar por las distintas dimensiones del árbol DOM

* + - En cierta medida análogo a la definición de rutas en un sistema de archivos Recomendación del W3C independiente del lenguaje de programación
    - Hay implementaciones en múltiples lenguajes
    - En particular, es uno de los principales elementos de XSLT Distingue entre 7 tipos de nodos:
    - Element
    - Diagrama

      Descripción generada automáticamenteAttribute
    - Text
    - Comment
    - Document
* Namespace
* Processing-instruction Relaciones entre nodos:
* Parent
* Children
* Siblings
* Ancestors
* Descendants

###### Ejemplos

* /bookstore/book[1]
* /bookstore/book[last()]
* /bookstore/book[last()-1]
* /bookstore/book[position()<3]
* //title[@lang]
* //title[@lang='en']
* /bookstore/book[price>35.00]
* /bookstore/book[price>35.00]/titleç

## XSL: EXTENSIBLE STYLESHEET LANGUAGE

CSS dice cómo mostrar un documento HTML

* HTML tiene un conjunto predefinido de etiquetas
* Decirle a un navegador cómo mostrar un elemento en una fuente o color especial es sencillo XML en cambio permite definir cualquier etiqueta
* ¿Qué significa TABLE?
* ¿Cómo debe mostrarlo el cliente?

XSL le dice al navegador cómo se debe mostrar el documento XML Una plantilla XSL es un documento XML

## EJEMPLO DE TRANSFORMACIÓN XSL

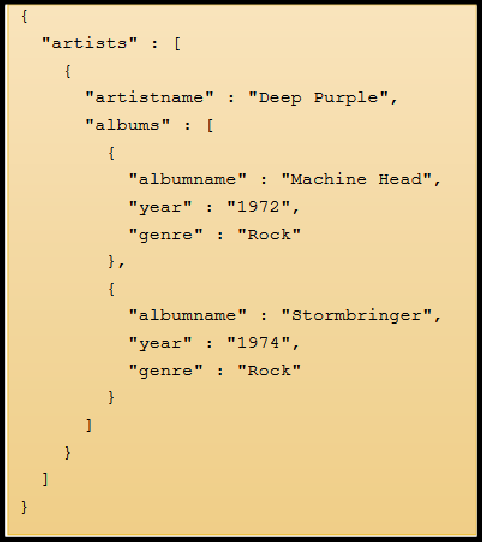
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**JSON: JAVASCRIPT OBJECT NOTATION**

* Según el W3C:
  + JSON is a syntax for storing and exchanging data
  + JSON is text, written with JavaScript object notation
* Muy popular: a día de hoy es seguramente el formato más utilizado para intercambiar datos entre aplicaciones web
* Es legible por personas e interpretable por algoritmos
* Almacena los datos siguiendo la notación de objetos y arrays JS

## SINTAXIS JSON

Un objeto JS es un conjunto desordenado de pares nombre/valor entre llaves ({})

* El nombre es una cadena de caracteres que identifica el atributo
* El valor puede ser un objeto, un array, un número, cadena de caracteres, true, false o null
* Un array es una colección ordenada de valores
* Permite construir listas de valores

## XML VS. JSON

En principio, XML y JSON sirven para lo mismo Sus diferencias son:

* JSON no utiliza tags
* JSON es más corto
* JSON es más rápido de leer y escribir
* JSON utiliza arrays
* XML debe ser parseado por un parser (tradicionalmente DOM), mientras que JSON puede serlo por el intérprete JavaScript como objetos y arrays JS “listos para utilizar”
* Los mecanismos de validación XML
  + https://json-schema.org/
  + Actualmente en versión draft

Escala de tiempo

Descripción generada automáticamente

# WEB INTERACTIVA. LAS APLICACIONES WEB

## WEB INTERACTIVA

El modelo Web hipertexto no permite más interacción del usuario que seguir hiperenlaces para obtener contenido estático en una lectura no secuencial de documentos

Diagrama

Descripción generada automáticamenteNecesario establecer comunicación entre programas que se ejecutan en el “servidor” con los datos que proporciona el usuario (del lado del cliente)

## FORMULARIOS HTML

Definidos mediante la etiqueta <FORM>

Contenedor que agrupa los elementos de entrada de datos, mezclados con otros elementos estándar HTML

Estructura:

<FORM ACTION=url METHOD=método de envío ...>

<INPUT> | <TEXTAREA> | <SELECT> | <BUTTON>

otros elementos HTML

</FORM>

* action: URL del recurso que se debe solicitar al enviar el formulario “Programa” que atiende la petición en lado servidor
* method: Método de envío de los datos asociados al formulario GET o POST

## CAMPOS DE FORMULARIO

Elementos HTML de interacción con el usuario de la aplicación:

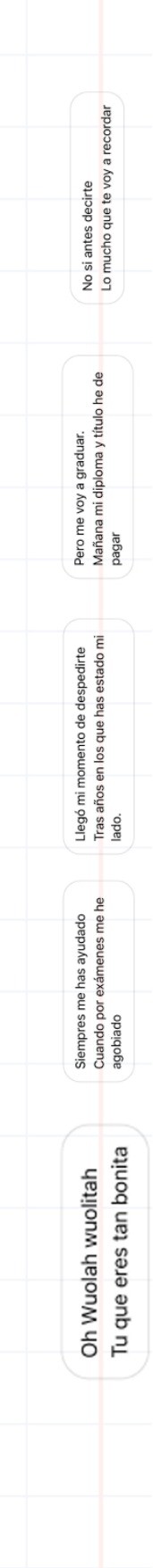
* INPUT: text, password, checkbox, radio, hidden, submit, reset
* En HTML 5 muchos más con sus pertinentes validaciones (p.ej., number o email)
* SELECT: Listas de selección con múltiples valores
* TEXTAREA: Campo de texto multilínea
* BUTTON: Botones de acción

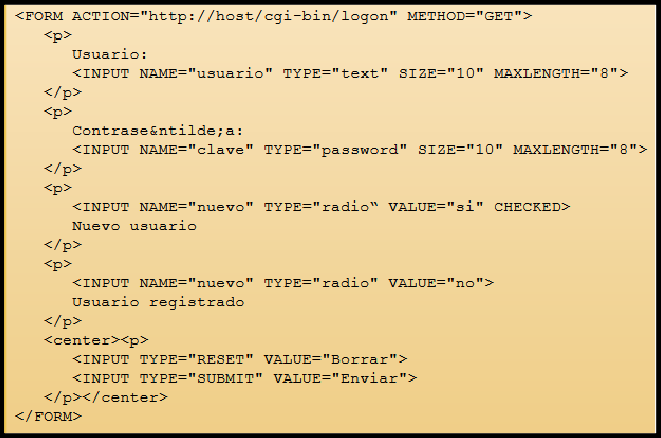
A cada campo de formulario se le asocia un nombre (atributo name) y un valor (atributo value)

La interacción con el usuario puede modificar el valor

Los resultados se envían al servidor en la petición HTTP mediante pares <nombre>=<valor>

* + Según el método elegido para el envío del formulario viajan asociados a la URL de procesamiento o en el cuerpo del mensaje HTTP



Ejemplo

### DIFERENCIA ENTRE GET Y POST

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

### PROCESAMIENTO DE FORMULARIOS EN EL SERVIDOR

La forma habitual de trabajar con formularios es usar un único “programa” que, dependiendo del método de envío, valide y procese el formulario o simplemente lo “muestre”

Ventajas:

* Simplifica el código y su mantenimiento
* Permite validar los datos del formulario en el propio formulario



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Procedimiento:

si petición POST (se ha enviado el formulario): si hay errores:

Mostrar formulario con errores si no:

Procesar formulario

si no (petición GET, no se ha enviado el formulario):

Mostrar formulario

fsi

## CGI: COMMON GATEWAY INTERFACE

CGI define un método “estándar” para que un servidor WWW pueda ejecutar programas externos y recoger información de ellos

* Es una extensión del protocolo HTTP
* El programa externo recibe del servidor información:
  + Asociada a la transmisión: Origen, URL, protocolo utilizado...
  + Introducida por el usuario, en un formulario de entrada.
* El paso de información se realiza mediante:
  + Variables de entorno
  + Línea de comandos
  + Entrada/salida estándar

## VARIABLES DE ENTORNO

**No dependientes de la petición:** SERVER\_SOFTWARE SERVER\_NAME GATEWAY\_INTERFACE

**Dependientes de la petición:** SERVER\_PROTOCOL SERVER\_PORT REQUEST\_METHOD PATH\_INFO PATH\_TRANSLATED SCRIPT\_NAME QUERY\_STRING REMOTE\_HOST

REMOTE\_ADDR

###### Asociadas a la seguridad de acceso:

AUTH\_TYPE REMOTE\_USER REMOTE\_IDENT

**Metainformación de la petición:** CONTENT\_TYPE CONTENT\_LENGTH

**Variables de la cabecera HTTP:** HTTP\_ACCEPT HTTP\_ACCEPT\_LANGUAGE HTTP\_USER\_AGENT HTTP\_COOKIE

## VENTAJAS E INCONVENIENTES

Ventajas:

* Sencillez de programación
* Uso de cualquier lenguaje de programación
* El programa CGI no afecta al funcionamiento del servidor por ejecutarse como un proceso independiente
* Estándar. Garantiza la portabilidad entre servidores de distintos fabricantes Inconvenientes:
* Lentitud
  + Cada ejecución requiere crear un proceso y finalizarlo, lo que implica reserva de memoria, apertura/cierre de ficheros, conexiones a bases de datos, etc.
* El programa CGI termina con cada llamada
  + No es posible el mantenimiento automático de un estado de la comunicación entre peticiones
  + La mayoría de aplicaciones requieren que el servidor mantenga información a lo largo de la sesión

## MANTENIMIENTO DE SESIÓN EN CGI

En los CGIs la gestión de sesiones está en manos de los programas Mediante campos de formulario ocultos:

* El usuario rellena un formulario con una serie de datos
* El cliente envía al servidor el formulario
* El programa CGI que lo recibe genera una página HTML en la que añade los valores recibidos como campos ocultos
* El usuario, al enviar nuevos formularios, proporciona al programa CGI los datos nuevos y los de la petición anterior

## SESIONES Y COOKIES

* Pequeño fragmento de información que el servidor (con el permiso del navegador) almacena en el cliente
* Cada vez que el navegador solicite una nueva página al servidor envía también la cookie
* Aunque en origen se crearon para la comunicación entre el cliente y el servidor, en la actualidad, la funcionalidad en el cliente también puede escribir y leer en las cookies
* Sobre el papel un gran invento. En la práctica, presentan serios problemas de seguridad

## PROBLEMAS DE SEGURIDAD DE LAS COOKIES

Aunque sobre el papel parecen un gran invento, en la práctica presentan serios problemas de seguridad:

* No todos los navegadores (usuarios) las aceptan ni gestionan correctamente
* Se almacenan de forma permanente en el navegador deben expirar
  + Si no se gestionan de forma correcta, un atacante puede usar el mismo navegador para suplantar la identidad de un usuario válido
  + Cross-site request forgery (CSRF)
* Comunicaciones: man in the middle, robo/modificación de cookies
  + HTTPS y aún así, evitar almacenar información sensible
* Privacidad y anonimato
  + Perfiles de usuarios a través de cookies de terceros
* Código en el cliente de terceros robo de cookies vía JS (XSS)
  + Cookies de tipo HTTPOnly

## WEB API: WEB APPLICATION PROGRAMMING INTERFACES

Surgen para tratar de evitar los problemas de bajo rendimiento de la interfaz CGI:

* Los nuevos programas se enlazan junto con el servidor en una librería dinámica
* El servidor llama a las funciones de librería como tareas dentro del propio proceso servidor
* El proceso servidor no finaliza: se mantienen ficheros abiertos, conexiones a bases de datos, etc. entre llamadas a funciones
* Se proporciona una API de acceso a los datos y estado del servidor Inconvenientes:
* Un fallo en una rutina puede hacer caer el servidor completo
* Lenguajes de programación normalmente limitados a C y C++ (en la actualidad más variedad)
* Difícil de programar. Es preciso conocer el funcionamiento interno del servidor para aprovecharla al máximo

Proporcionadas por casi todos los fabricantes: Netscape (NSAPI), Microsoft (ISAPI), IBM (ICAPI, GWAPI)...

## INTERFACES HÍBRIDAS

Intentan conseguir las ventajas de CGIs y Web APIs evitando sus inconvenientes

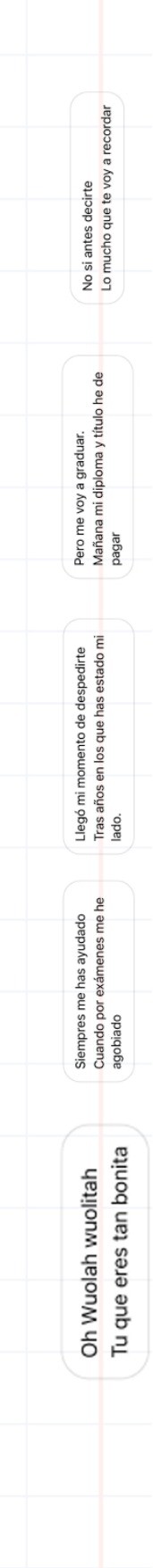
* Los programas se desarrollan de modo independiente al servidor Web, y en cualquier lenguaje
* El servidor Web, durante su inicialización, puede arrancar los programas en procesos independientes
  + Los programas arrancados, tras inicializarse, quedan a la espera de recibir peticiones
* Todo el proceso de inicialización se realiza antes de recibir una petición
* La comunicación mediante variables de entorno y stdin/stdout se sustituyen por un mecanismo de comunicación entre procesos más rápido
  + Puede permitir acceso remoto empleando un mecanismo de comunicación apropiado
  + Empleando los mismos elementos que en CGI se consigue facilidad de migración de programas CGI a las nuevas interfaces
* Tras atender una petición, el programa no finaliza: vuelve a esperar la siguiente petición
  + Es posible mantener el estado de la aplicación entre peticiones sucesivas
* Siguen este modelo:
  + FastCGI, de Open Market, Inc. Comunicación Servidor - Programas por Sockets
  + Netscape Web Application Interface (WAI). Comunicación mediante CORBA

## PÁGINAS DINÁMICAS

Como alternativa a CGIs y Web APIs, surgen extensiones del lenguaje HTML que permiten una mayor capacidad de procesamiento

En el cliente (client side scripting)

* Inclusión de código en el documento que el cliente interpretará para variar dinámicamente la presentación de la página
* Proporciona “inteligencia” en el navegador En el servidor (server side scripting):



* Inclusión de código funcional en el fichero HTML que contiene la descripción de la página
* El servidor lo interpretará para generar dinámicamente la página antes de su envío
* al cliente

### MODELOS DE IMPLEMENTACIÓN EN SERVIDOR

En PHP, por ejemplo, depende del servidor web asociado Con Apache hay dos modelos posibles:

* Multi-proceso: también llamado pre-forked, porque al iniciarse el servidor crea un pool de procesos que son reusados para atender las distintas peticiones
* Multi-hilo (workers): aunque el motor PHP está preparado para trabajar con múltiples hilos, hay muchas bibliotecas adicionales que no lo están

Modelo alternativo: asíncrono

Es el modelo implementado, por ejemplo, por NodeJS:

* Sistema para poder programa los servidores web en JavaScript
* Un hilo (y proceso, por supuesto)
* Llamadas no bloqueantes y programación basada en eventos
  + Por ejemplo, si quiero leer un fichero (para responder a petición http), invoco a la función de lectura, dándole el nombre de otra función (callback) y sigo ejecutando (posiblemente atendiendo otras peticiones)
  + Cuando se termina de leer el fichero (se produce el evento), el sistema invoca automática la función callback

### PHP: HYPERTEXT PREPROCESSOR

Lenguaje de programación de propósito general (interpretado) Sintaxis similar C/C++

Originalmente diseñado para el preprocesamiento de texto plano

En el contexto web utilizado para generar contenido dinámico del lado del servidor. Otros lenguajes similares son ASP o JSP

Los scripts PHP están incrustados en los documentos HTML y el servidor los interpreta antes de servir las páginas al cliente

El cliente no ve el código PHP, sino los resultados que produce Software abierto y gratuito

Integra acceso a formularios, archivos, bases de datos...

El acceso a la información propia del servidor se consigue a través de arrays indexados con el nombre del parámetro al que se desea acceder:

$\_SERVER: Variables propias del servidor

Ejemplos: $\_SERVER[QUERY\_STRING], $\_SERVER[REMOTE\_HOST]

$\_REQUEST: Variables asociadas a la petición recibida (campos de formularios)

$\_SESSION: Información asociada a la sesión de

* $\_COOKIE: Información asociada a las cookies
* ...



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

### EJEMPLO PHP

<html>

<body>

<?php echo "Hello World"; ?>

</body>

</html>

Vínculo URL – código

<http://www.misitioweb.com/index.php> Busca un fichero llamado ‘index.php’ y lo interpreta

**CONTENIDO ESTÁTICO VS. GENERADO DINÁMICAMENTE**

Diagrama

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

## RECEPCIÓN DE PARÁMETROS. LA FUNCIÓN $\_REQUEST

Interfaz de usuario gráfica

Descripción generada automáticamente

**RECEPCIÓN DE PARÁMETROS. LA FUNCIÓN $\_GET**

Res

Diagrama

Descripción generada automáticamente

## RECEPCIÓN DE PARÁMETROS. LA FUNCIÓN $\_POST

Diagrama

Descripción generada automáticamente

PHP está perdiendo peso pero sigue siendo importante.

## PYTHON PARA LA WEB

* Lenguaje de programación de propósito general (interpretado)
* Orientado a objetos
* En el contexto web utilizado para generar contenido dinámico del lado del servidor
* Las aplicaciones siempre se podrían programar desde cero, pero lo habitual es utilizar bibliotecas o frameworks
* El vínculo URL – código no es directo, depende de la biblioteca/framework que usemos
  + Lo habitual es realizar mapeos de nivel superior
  + Aquí vamos a ver Flask y Django

## FLASK

Micro-framework (en contraste con frameworks más completos como Django)

* Núcleo simple pero extensible
* No toma decisiones por el programador (como por ejemplo, tipo de base de datos)
* Provee librerías (extensiones) para, por ejemplo:
  + Integración de BD

Res

* + Validación de formularios
  + Gestión de ficheros (uploads)
  + Autenticación
* ... pero puedo ignorar todo esto e implementarlo por mi mismo

## MAPEO/ENRUTAMIENTO DE URLS

Establece las relaciones entre URLs y código Python (funciones)

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

## Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto Descripción generada automáticamenteMAPEO/ENRUTAMIENTO DINÁMICO

Texto

Descripción generada automáticamente

**MAPEO/ENRUTAMIENTO**

El mapeo puede ser dependiente del método utilizado para la petición HTTP

Texto

Descripción generada automáticamente

## VISTAS

Res

Las funciones como index() se convierten en manipuladores (handlers) de la URL asociada

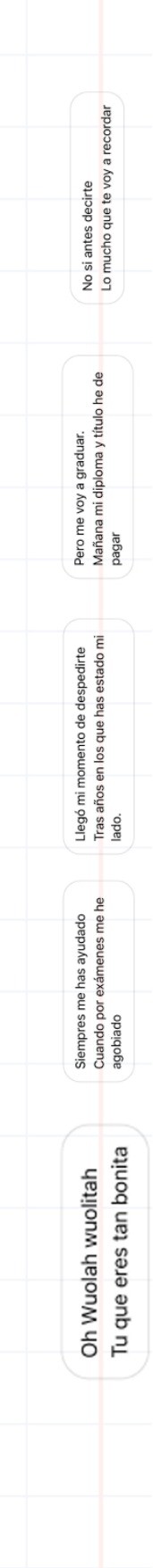
* En Flask (y otros frameworks) las funciones creadas para ser handlers se denominan vistas (views)
* Si basamos nuestra aplicación en una arquitectura MVC, los handlers suelen hacer las veces de controlador (cuidado con la posible confusión del nombre view)

En el return se puede poner directamente html entre comillas:

Texto, Carta

Descripción generada automáticamente

## Diagrama, Escala de tiempo Descripción generada automáticamentePara evitar que se complique: separación de responsabilidades



return '<!DOCTYPE html> ' \ '<html lang="es">' \ '<head>' \

…

<!DOCTYPE html>

<html lang="es">

<head>

<title> This is the page title </title>

</head>'

<body> <div id

="container">

<h1>Example of HTML Content</h1>

Return to the

homepage by

clicking <a href="/"> here</a>

</body>

</html>

### TEMPLATES

Para usar html dinámico.

Flask usa por defecto el motor (intérprete) Jinja2



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

## DJANGO

Framework de alto nivel para el desarrollo de aplicaciones web basadas en Python

* De mayor calado que Flask, sin llegar a ser un macro-framework
* Gestiona e integra prácticamente todos los aspectos relevantes del desarrollo de una aplicación web desde una perspectiva de proyecto cerrado
* La poca flexibilidad que permite en el diseño de la arquitectura software se consigue vía customización
* Paquetes necesarios para usar Django
  + Framework (django)
  + Herramientas para la gestión de proyectos Django (django-admin)

## DJANGO: LANZAR SERVIDOR HTTP

Como todo, a través del script de administración

$ python manage.py runserver

Dentro de la estructura de archivos que define el servidor con sus aplicaciones Lee los parámetros de configuración del fichero settings.py

## SESIONES

Gestionadas por un middleware que se debe activar en el fichero de configuración

* Permite abstraer al desarrollador de todo el problema del envío, recepción y gestión de datos

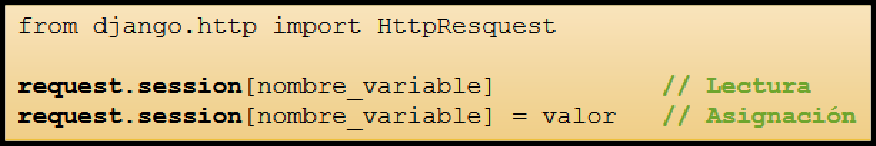
Por defecto, se basan en el almacenamiento de datos en el servidor y el envío de una cookie con un id de sesión

* También es posible un mecanismo basado en el almacenamiento de datos en cookies (menos recomendable)

Los datos de sesión por defecto se almacenan en la base de datos en unas tablas que se crean vía una migración

* También es posible configurarlo para que se almacenen en disco o en memoria

Cada vez que se recibe una nueva petición HTTP en el servidor, el middleware obtiene los datos de sesión del sistema de almacenamiento y los guarda en la Request en un objeto similar a un diccionario



Cuando se termina el procesamiento de una petición, el middleware persiste los datos almacenados en la Request

* Si han cambiado
* Se puede forzar la actualización:

Por defecto, los datos se serializan en JSON

## SERVIDOR WEB VS. SERVIDOR DE APLICACIONES

CGI

Server side scripting, incrustado en el documento web Servlet Mundo Java

WSGI Estándar Python

* Web Server Gateway Interface
  + Green Unicorn
  + mod\_wsgi

# EJECUCIÓN DE CÓDIGO EN CLIENTE

## OBJETIVOS

* + - Aumentar la capacidad de interacción del usuario con la aplicación
    - Disminuir el trasiego de datos entre el cliente y el servidor
      * Capturar los eventos generados por el usuario y responder a ellos sin depender del servidor
      * Realizar cálculos sencillos sin necesidad de comunicación con el servidor
      * Validar ciertos campos de formulario antes de enviarlos al servidor
    - Algunas tecnologías asociadas al uso de Javascript en el cliente:
      * DHTML = Dynamic HTML
        + HTML + JavaScript + CSS
        + No tiene nada que ver con el proceso de generación dinámica de documentos en el servidor (mediante Perl, PHP, JSP, ASP, etc.)
      * AJAX = Asynchronous JavaScript And XML
        + JavaScript + comunicación cliente-servidor asíncrona

Originalmente XML, actualmente más JSON

* + - * + Google Web Toolkit (GWT): framework creado por Google que permite ocultar la complejidad de aspectos de la tecnología AJAX
        + jQuery es otra biblioteca

## JAVASCRIPT

Originalmente propuesto por Netscape Communications Corporation en colaboración con Sun

* + - Estándar ECMA-262 Estándar ISO 12ª edición, junio 2021 https://[www.ecma-international.org/publications/standards/Ecma-262.htm](http://www.ecma-international.org/publications/standards/Ecma-262.htm)

En su uso típico para desarrollar funcionalidad en cliente, enriquece las páginas HTML incorporando características dinámicas e interactivas

* + - Interfaz de programación DOM (Document Object Model) Se ejecuta en cliente (en el navegador web)

Usos comunes (algunos ejemplos):

* + - Inclusión de efectos visuales en textos e imágenes de las páginas web
    - Manipulación de contenidos o aspecto de forma dinámica
    - Realización de operaciones matemáticas sencillas
    - Validación de datos introducidos en formularios
    - Gestión del sistema de navegación (menús desplegables)
    - Control del tipo y versión del navegador web, uso de la fecha y hora actuales, verificación de plugins...

Lenguaje de scripting interpretado Sintaxis muy similar a Java (y a C)

* + - Sensible a mayúsculas y minúsculas
    - Fin de sentencia con “;” o con salto de línea Orientado a objetos: objetos, propiedades, métodos, eventos
    - Peculiaridad: el concepto de clase no se introduce hasta versiones muy recientes Orientado a eventos

Es un lenguaje de programación

* + - Per se no proporciona los mecanismos para implementar funcionalidad a ejecutar en un cliente web HTML DOM

## HTML DOM

Cuando un navegador carga un documento HTML crea el modelo HTML DOM (Document Object Model) del documento

El modelo HTML DOM es un estándar W3C que define objetos para representar:

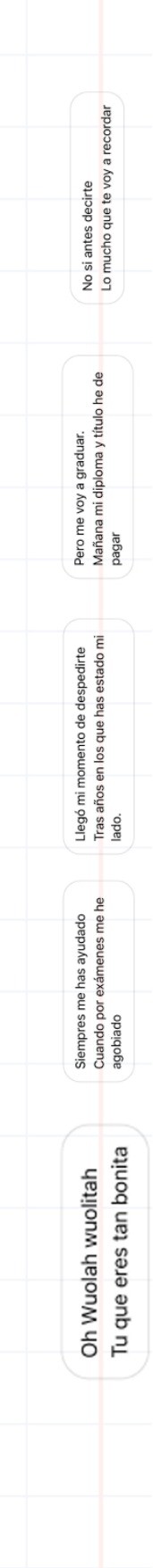
* + - Elementos HTML
    - Propiedades de los elementos HTML
    - Métodos para navegar por el árbol DOM
    - Eventos para capturar la interacción del usuario de la aplicación con los elementos HTML en la pantalla del navegador

Es la implementación del modelo HTML DOM en el navegador la que permite desarrollar funcionalidad DHTML

## JAVASCRIPT EN EL CLIENTE

Interpretado por el motor Javascript del navegador

* + - No se compila



* No se ejecuta de forma independiente como aplicación autónoma
* Se inserta en el código HTML y es interpretado por el navegador junto al resto del documento Enfocado a manejar elementos del documento y el navegador mediante objetos para acceder y manipular sus componentes.

Proporciona eventos para vincular funcionalidad a acciones del usuario.

### MODIFICAR EL ASPECTO VISUAL DEL DOCUMENTO EN JAVASCRIPT

Atributos style y className

document.getElementById("mi\_span").style.color = "#ff0000"; document.getElementById("mi\_span").style.backgroundColor = "#00ff00";

document.getElementById("mi\_span").className += " seleccionado";

Atributo classList

document.getElementById("mi\_span").classList.add("seleccionado");

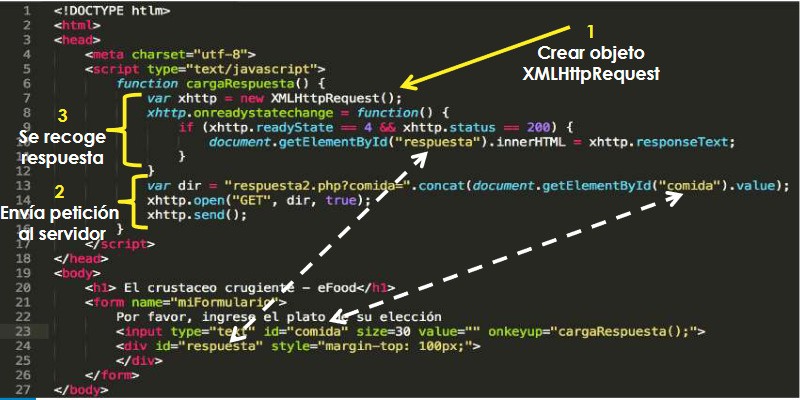
### AJAX

Asynchronous JavaScript and XML

* NO es un lenguaje de programación, sino una forma de usar tecnología existente en el momento en que se definió
* Conjunto de técnicas para intercambiar información con el servidor en background y actualizar partes de un documento HTML sin recargar la página completa
* Se basa en 4 estándares de Internet:
  + XMLHttpRequest object
  + JavaScript/DOM
  + CSS
  + XML
* Las aplicaciones AJAX no dependen del navegador ni de la plataforma! Ejemplo:



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.



## EL OBJETO XMLHTTPREQUEST

Se usa para realizar peticiones HTTP al servidor Soportado por todos los navegadores modernos

* + - Hace unos años era necesario comprobar si el navegador lo soportaba Envío de la petición HTTP:
    - open(tipo, url, asíncrona)
      * Configuración de la petición ≡ definición de un formulario
      * tipo: “GET” o “POST” ≡ form.method
      * url: destino de la petición ≡ form.action
      * asíncrona: true o false. Indicador de asincronismo!!!
    - send([query\_string])
      * Envía la petición (después de configurarla) ≡ submit
      * Petición GET: sin parámetros
      * Petición POST: recibe el query string (p.ej., "param1=valor1&param2=valor2")

xmlhttp.setRequestHeader("Content-type","application/x-www-form

-urlencoded")

Gestión del ciclo de vida de la petición:

* + - readyState
      * 0: la petición aún no se ha inicializado
      * 1: se ha establecido la conexión con el servidor
      * 2: se ha recibido la petición
      * 3: se está procesando la petición
      * 4: la petición ha terminado de procesarse y la respuesta está disponible Recepción de la respuesta:
    - Llamadas síncronas: después del send
    - Llamadas asíncronas: evento readystatechange
    - status: código de respuesta de la petición (p.ej. 200, 401, 403, 404 o 500)
    - responseXML: datos enviados por el servidor como isla de datos XML
    - responseText: datos enviados por el servidor en forma de string

## JQUERY

Biblioteca que trata de simplificar la programación de la lógica JavaScript en el lado del cliente:

* + - Acceso a la estructura del documento con una sintaxis similar a la de CSS
    - Fácil manejo de conjuntos de elementos
      * En las últimas versiones de JavaScript la diferencia no es muy significativa
    - Tratamiento específico de eventos
    - Comunicación con el servidor Debe ser importada:
    - Desde el propio servidor de la app web que estamos desarrollando
    - Desde un servidor público (CDN)

Es una biblioteca muy liviana que desde servidores rápidos se descarga muy rápidamente, agregando poca sobrecarga al tiempo de descarga de la página

Normalmente, una vez la página inicial es descarga, la biblioteca se cachea en alguna caché intermedia o la del propio navegador

Ejemplo:

## FILOSOFÍA JQUERY

Trabaja con estructuras similares a un array que permiten aplicar acciones sobre varios elementos en una misma sentencia

Unifica la sintaxis para prácticamente cualquier acción JavaScript:

1. Seleccionar el elemento o elementos HTML sobre los que queremos trabajar (de ahí el nombre jQuery)
2. Invocar a una función que realiza una acción sobre ellos
3. La función devuelve una lista de elementos se pueden enlazar llamadas, incluso aunque la lista no contenga ningún elemento

Sintaxis básica: $jQuery(selector).accion1().accion2()... Sintaxis abreviada: $(selector).accion1().accion2()...

* + - $: indica que lo que viene a continuación es una expresión jQuery
    - selector: idéntica los elementos HTML sobre los que trabajar reutilizando y ampliando la sintaxis de selectores CSS
    - accionX(): método jQuery al que invocar sobre los elementos seleccionados. Puede hacer referencia a la lectura/escritura de una propiedad del elemento, a un efecto visual, a una modificación DHTML, a la invocación o asignación de código de procesamiento de un eventos...

$("div#contenedor\_lista input:checked").parent().hide()

## ACCESO A LA ESTRUCTURA DEL DOCUMENTO

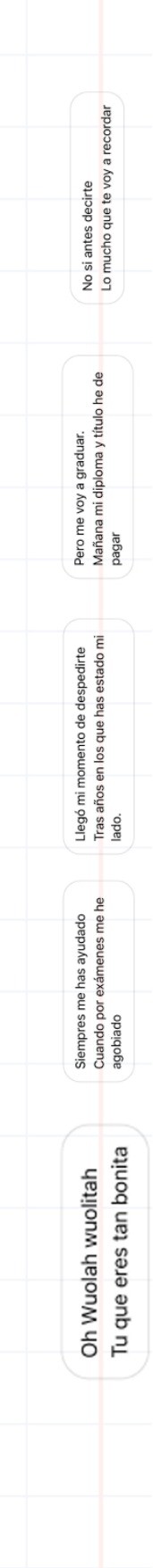
## Imagen que contiene Gráfico Descripción generada automáticamente

## RECORRIENDO EL ÁRBOL DOM

Funciones que facilitan recorrer la jerarquía. Ejemplos:

Texto, Carta

Descripción generada automáticamente



**ITERACIÓN SOBRE CONJUNTOS DE ELEMENTOS**

Como la selección y las funciones jQuery devuelven algo que imita a un array, se puede utilizar cualquiera de los bucles estándar proporcionados por JavaScript

jQuery ofrece además su propio bucle, más eficiente y con una sintaxis más Sencilla:

$('form :input').each(function(index, element) {})

* Los parámetros index y element de la función son, respectivamente, el índice y el valor de cada elemento sobre los que se itera

En las versiones más recientes de JavaScript los arrays incorporan un método forEach() análogo al de jQuery

### TRATAMIENTO DE EVENTOS

Texto

Descripción generada automáticamente con confianza baja



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

## MANIPULACIÓN DINÁMICA DEL DOCUMENTO HTML

Tabla

Descripción generada automáticamente

## MANIPULACIÓN DINÁMICA DEL DOCUMENTO HTML

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

Supongamos que queremos agregar un \* rojo a la izquierda de cada input “required” del documento



Esto se puede conseguir con una única sentencia jQuery:

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza baja

## PETICIONES AL SERVIDOR

ajax({parametro:valor, parametro:valor, ... })

* + - url: url de la petición
    - async: el valor por defecto es true
    - type: "GET" o "POST“
    - data: datos que se van a mandar al servidor asociados a la petición
    - contentType: el valor por defecto es "application/x-www-form-urlencoded"
    - dataType: tipo de datos esperado en la respuesta (p.ej., "xml", "html", "json")
    - context: indica cuál va a ser el valor de this para las funciones callback
    - success: función o array de funciones callback que ejecutar cuando la petición finaliza de forma satisfactoria
    - error: función o array de funciones callback que ejecutar en caso de error en la petición
    - timeout: timeout de la petición expresado en milisegundos get(url [,data] [,success] [,dataType])

getJSON(url [, data] [, success]) getJSON(url [, data] [, success]) getScript(url [, success])

post(url [, data] [, success] [, dataType] ) load(url [, data] [, complete])

## COMENTARIO FINAL SOBRE FRAMEWORKS

“Real-world programming” rara vez se hace completamente desde 0

* + - Sólo con HTML + CSS + JS “desnudos”

Existen múltiples frameworks (e incluso CMSs) que ayudan en la tarea Dos aspectos importantes:

* + - Ninguna herramienta es la “bala de plata” que va a solucionar todos los problemas del mundo
    - Mientras que la tecnología base es estable a medio plazo (HTML 1991, CSS 1996, JS 1995), los frameworks y tecnología similar son mucho más efímeros y cambiantes

Moraleja: poner especial cuidado a la hora de elegir un framework o biblioteca de apoyo al desarrollo

## ALGUNOS FRAMEWORKS POPULARES

Boostrap

* + - Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web

ZURB Foundation

* + - The most advanced responsive front-end framework in the world HTML5 Boilerplate
    - The web’s most popular front-end template Angular (AngularJS)
    - HTML enhanced for web apps! ReactJS
    - A JavaScript library for building user interfaces

## DESARROLLOS INTERESANTES

###### Meteor

https://[www.meteor.com/](http://www.meteor.com/)

“Meteor is a complete open source platform for building web and mobile apps in pure JavaScript” Claves:

El mismo código se va a ejecutar en todos los clientes (navegadores y dispositivos móviles) y en el servidor

Cambios en los datos se reflejan inmediatamente en todas las visualizaciones

Un comando envía la app a producción y actualiza todos los dispositivos conectados (sin necesidad de pasar por la app store de turno)

# HTML5

## ¿QUÉ ES HTML5?

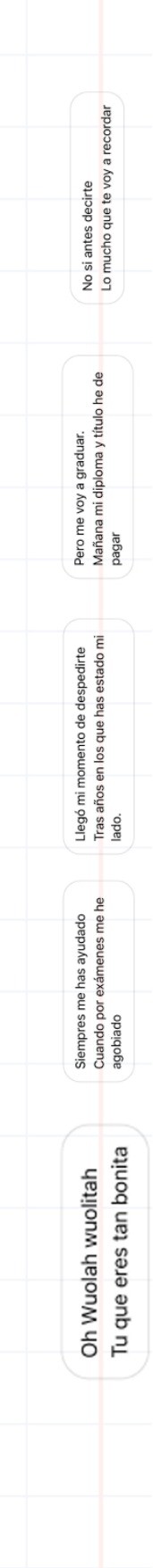
HTML 5 = HTML + CSS + JavaScript

* + - Es el nuevo estándar para HTML, XHTML y HTML DOM
    - Estándar W3C desde octubre 2014
    - Reglas para su especificación:
      * Toda característica nueva debe basarse en HTML, CSS, DOM y JavaScript
      * Reducir la necesidad de plugins externos (p.ej.: Flash)
      * Mejor manejo de errores
      * Más etiquetas para evitar tanto scripting
      * Independiente del dispositivo

## CARACTERÍSTICAS ADAPTADAS A LAS NUEVAS NECESIDADES

* + - Nuevos elementos específicos sobre estructura del documento:
      * header, nav, article, section, aside, footer
    - Nuevos controles para los formularios
      * date, time, email, url, search...
    - Tag para video y audio
    - Canvas (lienzo) para dibujar
    - Uso de SVG: crear gráficos vectoriales
    - Mejor soporte al almacenamiento local offline: SQLite
    - Interfaz de usuario gráfica, Aplicación, Gráfico de rectángulos

      Descripción generada automáticamenteAPI para geolocalización para JavaScript que permite obtener la localización del usuario si éste (y su dispositivo) lo permite
    - Web storage



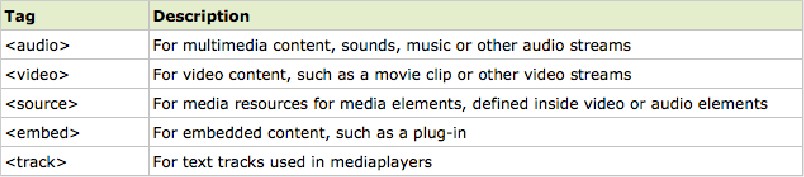
Tabla

Descripción generada automáticamente ALGUNAS DE

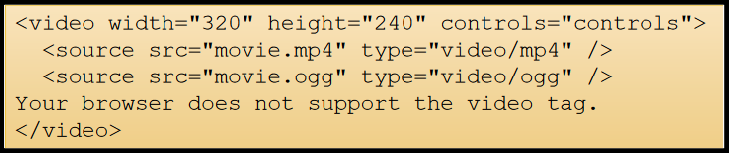
LAS NUEVAS

ETIQUETAS

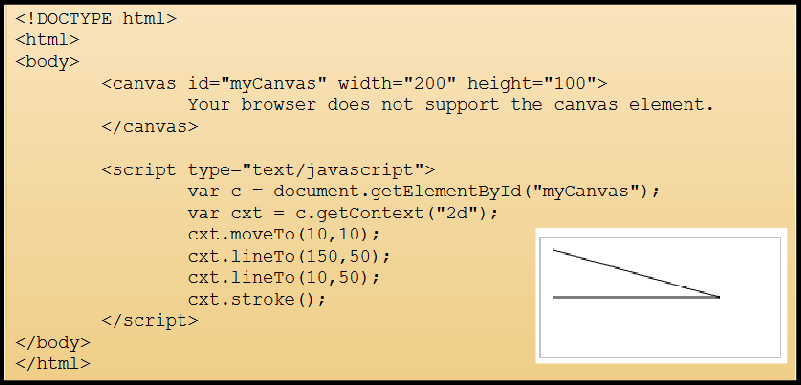
Nuevos elementos multimedia:



Ejemplo vídeo



Ejemplo canvas

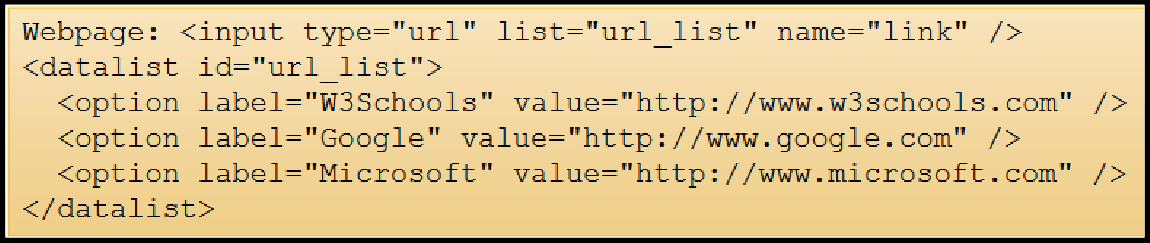


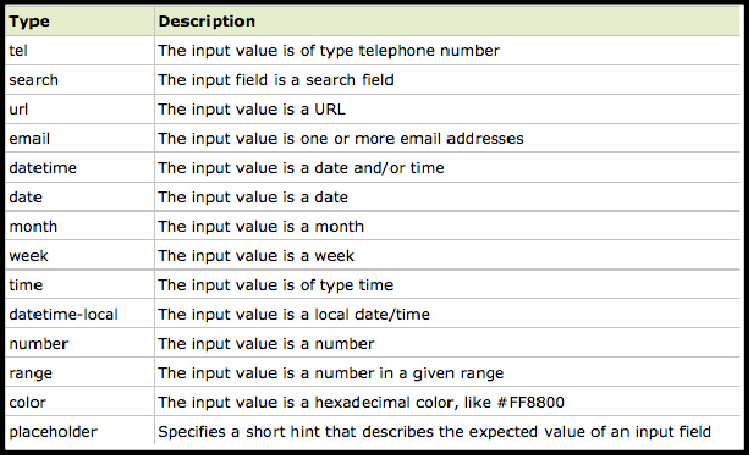


Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

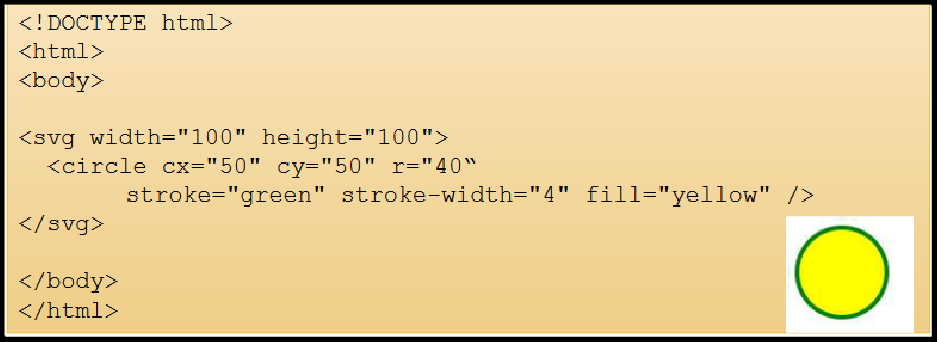
Descripción generada automáticamenteNuevos campos formulario

Ejemplo datalist



Nuevos tipos de inputs

SVG (SCALABLE VECTOR GRAPHIC)

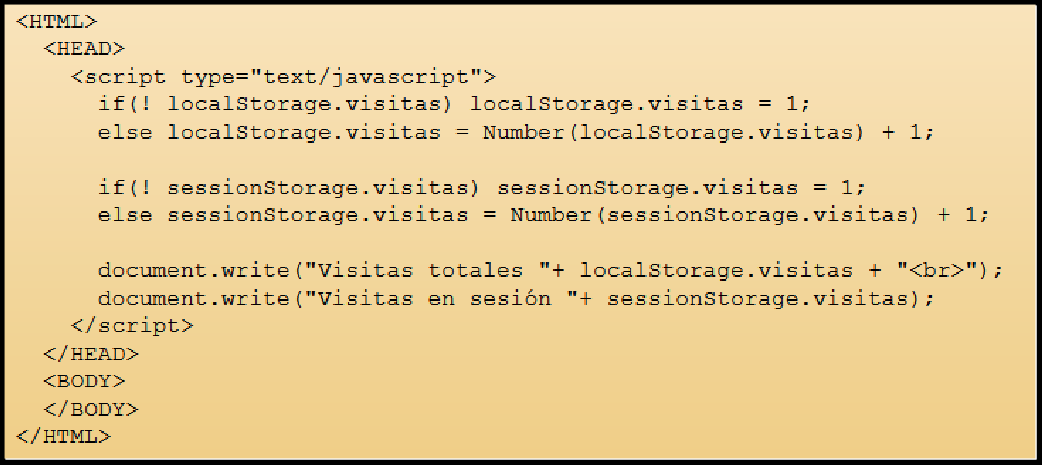


## LOCALSTORAGE Y SESSIONSTORAGE

Almacenes de datos en window

Almacena pares clave/valor: setItem, getItem, removeItem y clear Permiten el acceso mediante sintaxis abreviada

Res



# APLICACIONES WEB. ALTERNATIVAS DEL LADO DEL SERVIDOR

## CAMBIO EN LAS NECESIDADES

Desde el origen de la WWW la tecnología se ha ido adaptando a las nuevas necesidades:

* + - Ya no quiero servir documentos que leer de manera no secuencial
    - Quiero ejecutar aplicaciones

En temas anteriores nos hemos focalizado en PHP, Python y JavaScript como tecnologías para ejecutar funcionalidad en el servidor, pero existen otras muchas alternativas

## JAVA EE: JAVA PLATFORM, ENTERPRISE EDITION

Extensión de la Java SE con especificaciones y APIs para:

* + - Gestión de transacciones
    - Acceso a base de datos
    - Desarrollo de aplicaciones web
    - Etc

Objetivo en el contexto WWW: ejecutar componentes Java en el servidor Filosofía: lenguaje único sobre múltiples plataformas (cf. .Net)

Interfaz con los elementos de comunicación establecidos en el protocolo HTTP y los formularios HTML Especificaciones web:

* + - Programas en el servidor: Servlets
    - Páginas dinámicas: JavaServer Pages, JSP
    - Enterprise Java Beans, EJB
    - Java Server Faces, JSF
    - Distintas APIs para web services
    - ...

## SERVLETS

Especificación de más bajo nivel

Extienden la funcionalidad de los servidores web con una arquitectura basada en componentes “ejecutables”

Aplicación Java que se ejecuta en el servidor gestionando y procesando peticiones HTTP

* + - Se ejecutan totalmente en el servidor bajo petición de un cliente (vs. applets)
    - Se invocan a través de una URL que identifica el programa
    - Reemplaza a los programas de interfaz CGI. Sintaxis más sencilla
    - Necesitan un servidor JEE específico: Tomcat, JBOSS, GlassFish... Su implementación se incluyó en Java EE 5 SDK: javax.servlet.\*

Todo servlet debe implementar la interfaz javax.servlet.Servlet

* + - javax.servletGenericServlet
    - javax.servlet.http.HttpServlet

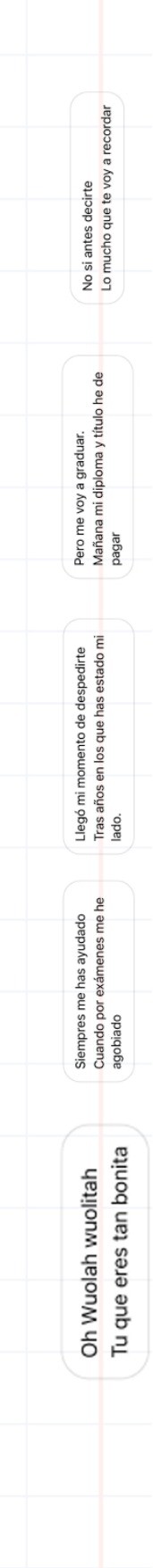
## VENTAJAS DE LOS SERVLETS

Portabilidad y flexibilidad:

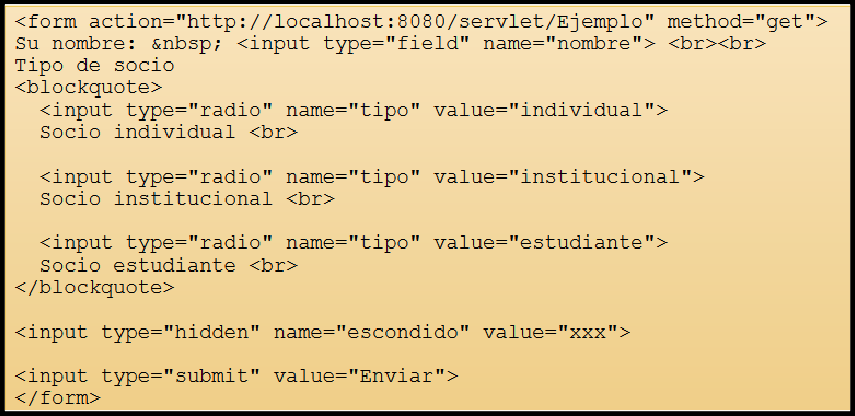
* + - La conexión del servlet se basa en una API independiente de la plataforma definida en el estándar de Java
    - El código Java altamente transportable
    - Permite aprovechar objetos reutilizables (EJBs) Seguridad:
    - Los servlets se ejecutan bajo un único proceso (servlet engine)
    - Permite acceso protegido integrado en un entorno de autenticación única Rendimiento:
    - Entorno de ejecución propio
    - Se ejecutan y permanecen en memoria
    - Se pueden precargar o cargar bajo demanda
    - Mantienen sesiones entre peticiones HTTP
    - Son multitarea
    - Escalables entre multiprocesadores y sistemas heterogéneos

Diagrama

Descripción generada automáticamente



Ejemplo parte cliente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

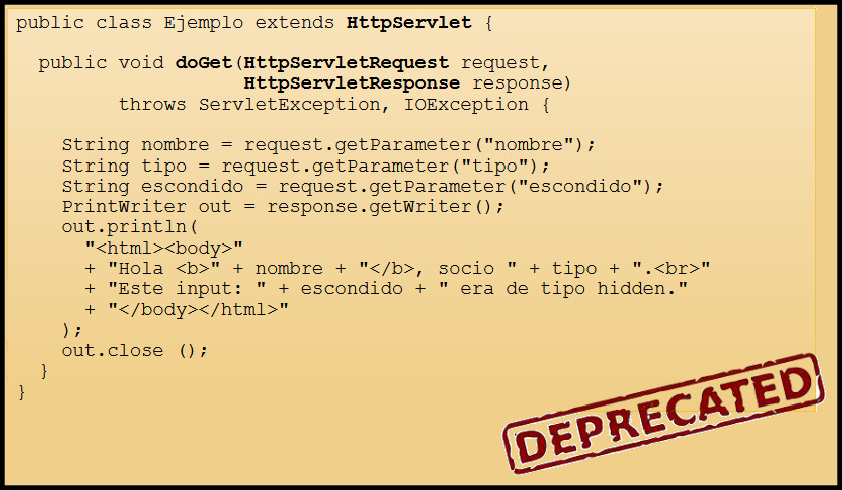
Ejemplo servidor

Imagen que contiene Rectángulo

Descripción generada automáticamente



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

### IMPLEMENTACIÓN DE SESIONES

Interfaz javax.servlet.http.HttpSession

* Id de sesión
* Fecha de creación
* Fecha de último acceso
* Mecanismos para invalidar la sesión

Aunque las implementaciones de JEE lo hacen automáticamente, no hay “magia” que modifique el protocolo HTTP (protocolo sin estado)

* Cookies
* Reescritura de URL
* Campos ocultos

Con las limitaciones y ventajas de cada uno de ellos

## JSP: JAVASERVER PAGES

Tecnología Java de server side scripting Generación dinámica de contenidos

Un archivo JSP contiene:

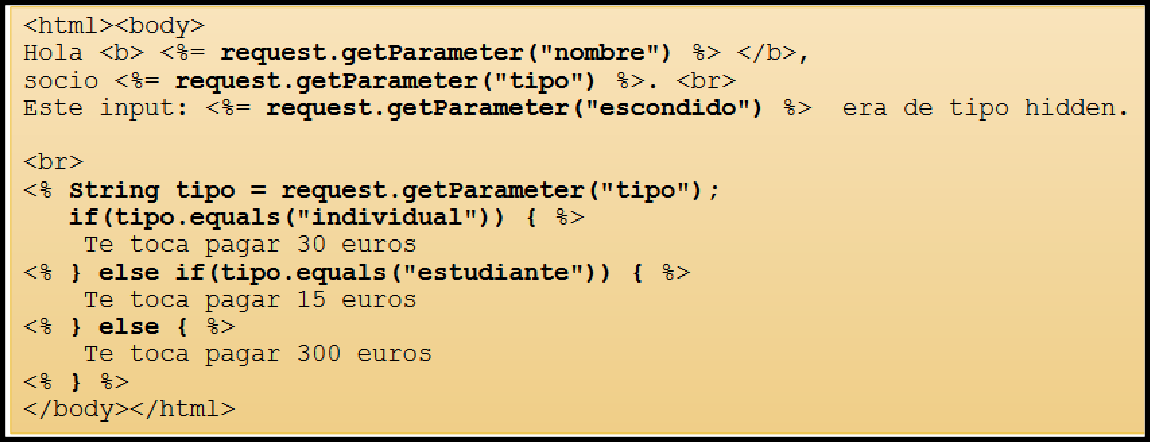
* + - Código HTML nativo (template data)
    - Elementos JSP: directivas y acciones
    - Código funcional Java
      * Se pueden utilizar variables implícitas: request, response...
    - Mecanismos de extensión de etiquetas: JSP Standard Tag Library, JSTL Necesita un servidor web que soporte JSP

Se solicitan igual que un documento HTML desde un navegador

## CICLO DE INVOCACIÓN DE JSP

Se ejecutan en un servlet engine específico (jsp engine):

* + - Convertidas en un servlet dinámicamente la primera vez que se ejecutan
    - Se crea un objeto que implementa la interfaz HttpJspPage

Ejemplo

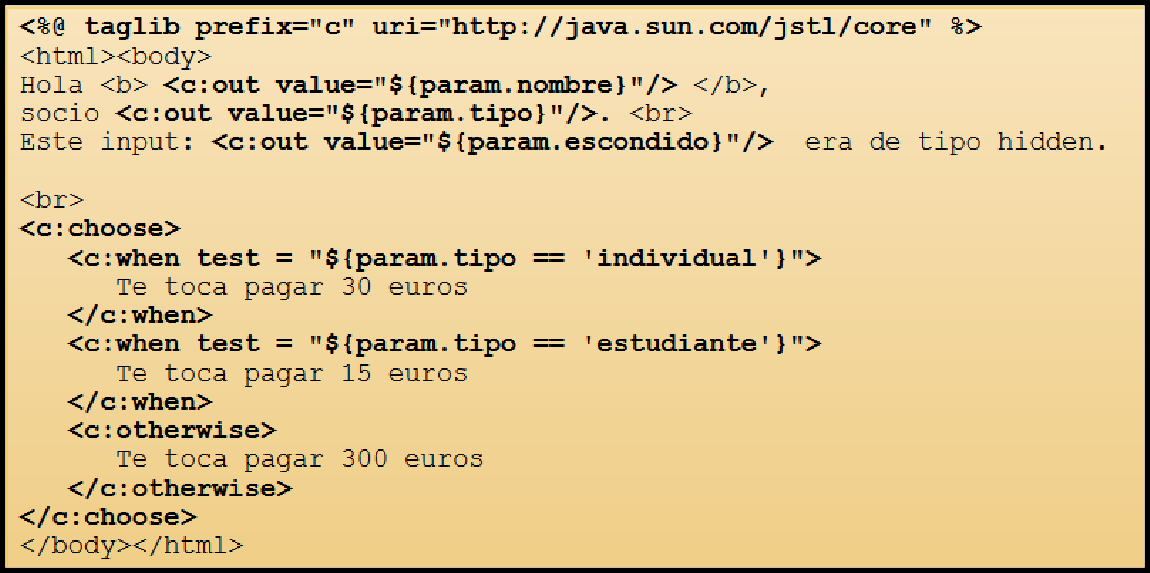
## BIBLIOTECAS DE ETIQUETAS

El código se vuelve farragoso y surgen bibliotecas de etiquetas para separar código funcional del HTML

* + - Específicas
    - Estándar: JSP Standard Tag Library, JSTL Las bibliotecas englobadas en JSTL son:
    - core: iteraciones, condicionales, manipulación de URL y otras funciones generales.
    - xml: para la manipulación de XML y para XML-Transformation.
    - sql: para gestionar conexiones a bases de datos.

Res

* + - fmt: para la internacionalización y formateo de las cadenas de caracteres como cifras



## EJB: ENTERPRISE JAVA BEANS

Desarrollada originalmente por IBM

API Java para encapsular lógica de negocio en back-end

* + - Persistencia, transaccionalidad, planificación, seguridad... EJB sin estado vs. EJB con estado

## ARQUITECTURA .NET

Arquitectura de Microsoft para aplicaciones Web Consta de tres componentes principales:

* + - Un entorno de aplicaciones independiente del lenguaje y optimizado para el entorno distribuido:

.NET Framework

* + - Un entorno de desarrollo para la programación de aplicaciones: Visual Studio .NET
    - Un sistema operativo que soporta entornos distribuidos y el framework .NET: Windows Server Filosofía: plataforma única compartida por múltiples lenguajes
    - Compilados a Microsoft Intermediate Language (MSIL)

Interfaz de usuario gráfica, Texto, Aplicación, Tabla

Descripción generada automáticamente

## .NET FRAMEWORK

* + - Common Language Runtime, CLR: Entorno de ejecución de los programas MSIL
      * Compilación Just In Time (JIT) al entorno de ejecución
    - Base Class Library: Biblioteca básica de funciones estándar, como las de gestión de E/S, seguridad, comunicaciones, tareas, texto...
    - ADO.NET: Clases para acceder a datos a través de ODBC, OLE DB, Oracle o SQL Server
    - XML: Clases para manipulación de documentos XML
    - ASP.NET: Clases para el soporte de construcción de páginas Web
      * Soporta formularios Web y Web Services
    - Windows Forms: Clases para soportar la construcción de clientes Windows
    - .NET Framework no integra componentes para realizar lógica de negocio similares a EJBs. Se deben realizar como servicios de Windows o como componentes COM+

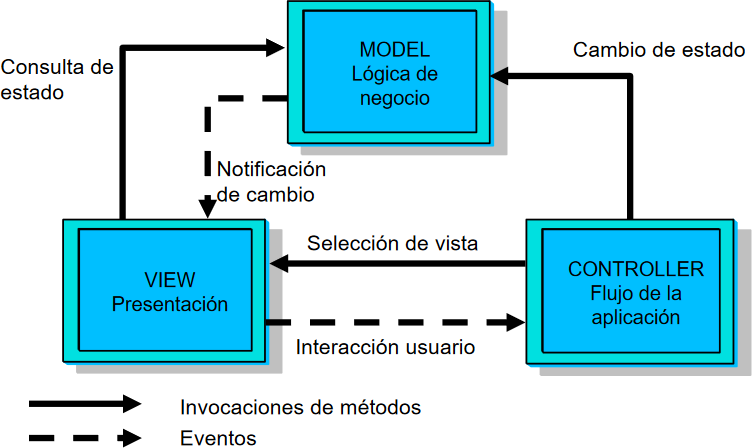
## MODELO DE ARQUITECTURA DE APLICACIONES WEB

Algunos de los requerimientos de las aplicaciones Web actuales son difíciles de resolver:

* + - Soporte de múltiples canales de presentación: clientes HTML, Java, WAP, portales de voz...
    - La funcionalidad a presentar al cliente depende del perfil del mismo
      * Puede necesitarse personalización
    - Los ciclos de desarrollo y puesta en producción son cortos
    - Hay gran cantidad de aplicaciones heredadas (legacy) ya desarrolladas

Un modelo de arquitectura de aplicaciones que intenta resolver estos problemas es el Modelo-Vista-Controlador (MVC)

* + - Recomendado para desarrollo en la JEE

**MVC**

Modelo:

* + - Representa los datos de la aplicación y las funciones de negocio
    - Los encapsula para hacer transparente su manejo al resto de la aplicación
    - Permite tener la funcionalidad de la aplicación independiente, y en cualquier tipo de soporte, aunque sean sistemas heredados

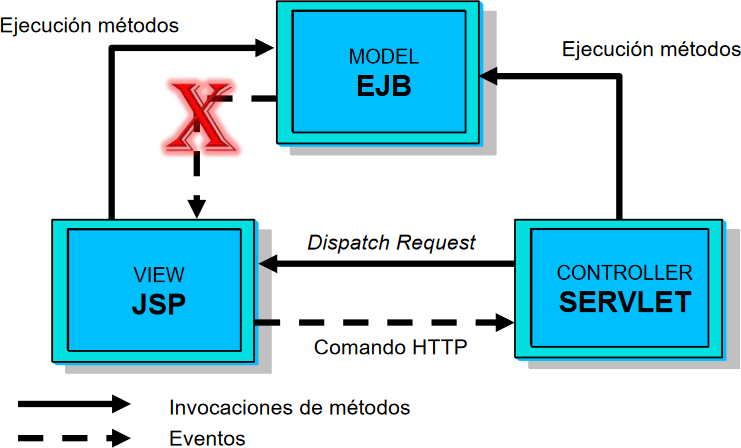
Vista:

* + - Realiza la presentación de los datos del modelo
    - Accede al modelo y adapta los datos presentados al cliente final
    - Permite múltiples vistas dependiendo del medio de salida de los datos, sin necesidad de cambiar la lógica de la aplicación ni la navegación de la misma

Controlador:

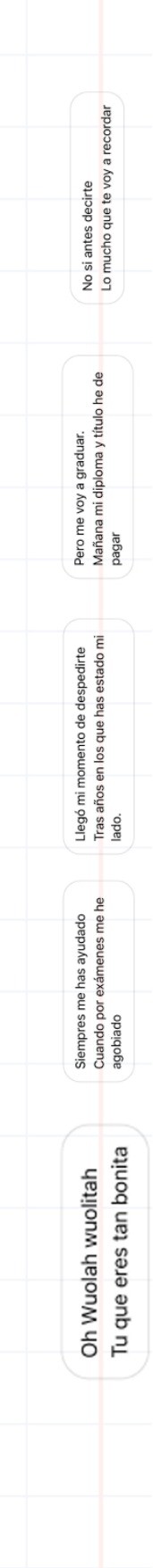
* + - Recibe las interacciones del usuario y las traslada en acciones sobre el modelo
    - Decide la salida que es necesario presentar al usuario solicitándolo a la Vista
    - Diagrama

      Descripción generada automáticamentePermite realizar la navegación que se necesite para el usuario sin necesidad de alterar la vista ni el modelo



MVC EN JEE

ARQUITECTURA MULTINIVEL MVC





Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.