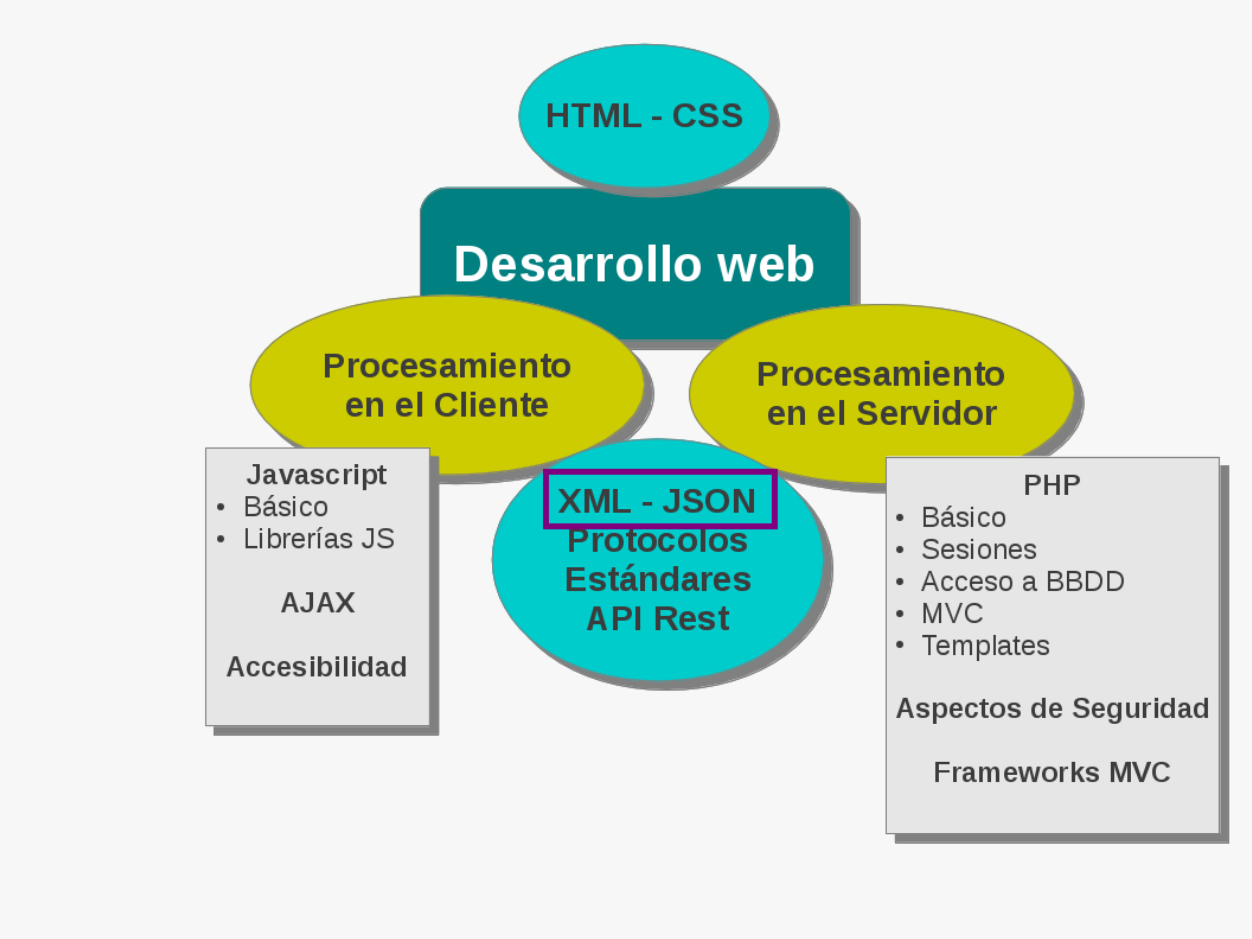
title: Clase 6 Proyecto 2014
Author: Einar Lanfranco, Claudia Banchoff
description: Notaciones para descripción de datos
keywords: XML + JSON + YAML
css: proyecto.css

Proyecto de Software

Cursada 2014

Hoy seguimos con ...



Temario

- **Repaso Clase Anterior**

    - **Consultas a las BBDD**
        - SQL Injection
    - **MVC**
        - la vista con templates
- **Describiendo información**
    - XML
    - JSON
    - YAML

Repaso - Inyección SQL

- Una inyección SQL suele ocurrir cuando se arma en forma descuidada una consulta a la base de datos a partir de los datos ingresados por el usuario.
- Dentro de estos parámetros pueden venir el código malicioso.
- Ejemplos típicos:

```
select * from users where id='". "1' or '1=1" ."' and
                          pass='". "1' or '1=1" ."'";
```

- Lo que se se resuelve en:

```
select * from users where id='1' or '1=1'
                    and pass='1' or '1=1';
```

Repaso - Prepared Statement

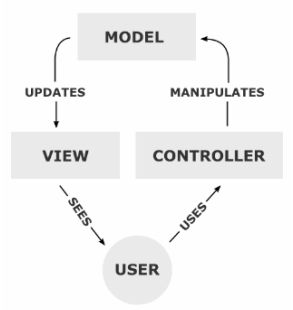
- Pueden definirse como un tipo de plantillas compiladas para SQL que las aplicaciones quieren ejecutar, pueden ser personalizadas usando parámetros de variables.
- Ejemplo:

```
<?php
   $db_host="127.0.0.1";
   $db_user="user";
   $db_pass="pass";
   $db_base="base";

   $cn = new PDO("mysql:dbname=$db_base;host=$db_host",$db_user,$db_pass);

   $query = $cn->prepare("SELECT * FROM usuarios where nombre=? and pass=?");
   $query->execute(array($_POST["email"],$_POST["pass"]));
?>
```

Repaso - MVC



- Reduce la complejidad, facilita la reutilización y acelera el proceso de comunicación entre capas.

Repaso - MVC

```php
<?php
//Conectamos a la Base
require_once("2-modelo.php");

//Recupero los usuarios
$usuarios=obtener_usuarios();

//Cargo la vista
require("2-vista.php");

?>
```

```php
<?php
function obtener_usuarios(){
    $db_host='127.0.0.1';
    $db_user='user';
    $db_pass='pass';
    $db_base='base';
    $link = mysqli_connect($db_host,$db_user,$db_pass,$db_base) o
    $resu=$link->query('select * from usuarios');

    while ($dato = mysqli_fetch_array($resu))  {
        $usuarios[]=$dato;
    }
    // Cierro la conexión
    mysqli_close($link);
    return $usuarios;
}
?>
```

```php
<html>
<head><title>Listados de Usuarios</title></head>
<body><h1>Listado de Usuarios Activos </h1>
<table>
<tr><th>Nombre</th><th>DNI</th></tr>

<?php foreach ($usuarios as $usuario){
    echo "<tr><td>".$usuario->nombre . "</td>";
    echo "<td>". $usuario->dni."</td></tr>";
};?>

</table>
</body>
</html>
```

class: destacado

Repaso - Templates

- El uso de templates o plantillas permite separar la aplicación de la presentación, pero ....

No asegura MVC. **Esa es NUESTRA responsabilidad**

Repaso - Twig

- Los templates **se utilizan para definir la vista**.
- Tienen un formato especial.
- No utiliza una extensión en particular (podría ser html, xml, twig, tpl, etc.).
- Son procesados por el sistema de plantillas.
- Contienen variables o expresiones que son reemplazadas cuando se procesa el template y tags que proveen la lógica de la presentación.
- Ejemplo: template-con-twig

Describiendo información

XML, JSON, YAML ....

XML - eXtensible Markup Language

- Es un lenguaje de marcas.
- Es un metalenguaje.
- Surge de la necesidad de contar con un mecanismo para describir información estructurada.
- XML describe semántica.
- **Existe SGML – "Standarized General Markup Language", pero ...**
    - Es complejo de procesar.
- **Existe HTML, pero ...**
    - NO fue pensado para este fin.

XML - Sintaxis Básica

¿Nos suena conocido?

```
<?xml version="1.0"?>
<ficha>
        <nombre>Roland Garros</nombre>
        <lugar>Paris</lugar>
        <estadioPrincipal fotoEstadio="estadio.jpeg"/>
</ficha>
```

- Usamos etiquetas, aunque las definimos nosotros...

XML - Sintaxis Básica

Misma sintaxis de HTML, aunque con algunas consideraciones:

- Elementos: Vacíos y No vacíos.
- Existe sensibilidad a mayúsculas y minúsculas.
- Debe tener una **única** raíz.
- **Atributos asociados a la etiqueta de apertura.**
    - pares **nombreAtributo="valorAtributo"**
    - SIEMPRE entre **""**

Espacios de Nombres

- Permiten separar e identificar elementos duplicados.
- Ejemplo1:

```
<?xml version="1.0"?>
<ficha>
        <nombre>Roland Garros</nombre>
        <lugar>Paris</lugar>
</ficha>
```

- Ejemplo2:

```
<?xml version="1.0"?>
<ficha>
        <elemento>Plata</elemento>
        <simbolo>Ag</simbolo>
</ficha>
```

data-scale: 0.7

Conflicto de nombres: ¿Cómo se resuelve?

- Definiendo un prefijo
- Ejemplo1:

```
<?xml version="1.0"?>
<Torneos:ficha>
        <Torneos:nombre>Roland Garros</nombre>
        <Torneos:lugar>Paris</lugar>
</Torneos:ficha>
```

- Ejemplo2:

```
<?xml version="1.0"?>
<TbPeriodica:ficha>
        <TbPeriodica:elemento>Plata</elemento>
        <TbPeriodica:simbolo>Ag</simbolo>
</TbPeriodica:ficha>
```

Definiendo espacios de nombres

- Se utiliza el **atributo xmlns**:
- Son opcionales.

```
<Torneos:ficha  xmlns:Torneos="http://www.miServidor.com/ejemplo">
        <Torneos:nombre>Roland Garros</nombre>
        <Torneos:lugar>Paris</lugar>
        ....
</Torneos:ficha>
....
<TbPeriodica:ficha xmlns:TbPeriodica="http://www.miServidor.com/otro">
        <TbPeriodica:elemento>Plata</elemento>
        <TbPeriodica:simbolo>Ag</simbolo>
                ....
</TbPeriodica:ficha>
```

Definiendo espacios de nombres

- Otra forma ...

```
<raiz
xmlns:Torneos="http://www.miServidor.com/ejemplo"
xmlns:TbPeriodica="http://www.miServidor.com/otro">

<Torneos:ficha>
        <Torneos:nombre>Roland Garros</nombre>
        ....
</Torneos:ficha>
....
<TbPeriodica:ficha>
        <TbPeriodica:elemento>Plata</elemento>
                ....
</TbPeriodica:ficha>
</raiz>
```

Espacios de nombres: ¿Cuándo usar?

- Si se tiene que compartir con OTRO documento que pueda contener elementos con los mismos nombres.
- Se utilizan para identificar tipos de datos o elementos asociados a otras tecnologías XML.

Espacios de nombres: ¿Cuándo usar?

- Ejemplo de XSL (hojas de estilo para documentos XML):

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html> <body>
        <h1>Saludos!!!</h1>
        <p> Mis Amigos hasta ahora...</p>
        <ul>
          <xsl:for-each select="amigos/amigo">
                <li><xsl:value-of select="nombre"/></li>
          </xsl:for-each>
        </ul>
  </body>  </html>
</xsl:template>
</xsl:stylesheet>
```

¿Cómo validar un documento XML?

Representan lo mismo, pero...

```
<ficha>
   <nombre>Roland Garros</nombre>
   <fechaInicio>22/05/2011</fechaInicio>
   <estadioPrincipal>
      <archivoFuente>estadio.jpeg</archivoFuente>
   </estadioPrincipal>
</ficha>
```

```
<ficha>
  <nombre>Roland Garros</nombre>
  <fechaInicio dia="25" mes="05" anio="2011" />
  <estadioPrincipal archivo="estadio.jpeg" />
</ficha>
```

```
<ficha>
   <nombre>Roland Garros</nombre>
   <fechaInicio>
      <dia>25</dia>
      <mes>05</mes>
      <anio>2011</anio>
   </fechaInicio>
   <estadioPrincipal>
      <archivoFuente>estadio.jpeg</archivoFuente>
   </estadioPrincipal>
</ficha>
```

DTD – Document Type Definition

- Describe la “gramática” del documento.
- **Define los elementos del documento XML:**
  - Qué elementos.
  - Qué atributos.
- Elementos vs. atributos

¿Cómo se asocia un DTD a un documento XML?

- Si se encuentra en archivo externo:

```
<!DOCTYPE elementoRaiz SYSTEM “http://servidor/DTD/archi.dtd”>
```

- Puede incluirse en la definición del documento:

```
<!DOCTYPE elementoRaiz[
        definiciones
]>
```

Recordamos de la primer clase ...

```
<!DOCTYPE html  PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<!DOCTYPE html
         PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!DOCTYPE html>
```

- Veamos el ejemplo: xhtml1-strict.dtd

DTD: ¿Cómo se define?

```
<?xml version="1.0"?>
<!DOCTYPE raizDelArbol [
<!ELEMENT  .......>
<!ATTLIST  .......>
<!ENTITY  .......>
]>
<raizDelArbol>
        <elemento>......</elemento>
        <elementoVacio/>
..... &entidad;
<!-- Comentarios -->
</raizDelArbol>
```

Seguimos con el tenis...

```
<?xml version="1.0"?>
<!DOCTYPE ficha [
<!ELEMENT  ficha (nombre+, lugar+, estadioprincipal?)>
<!ELEMENT  nombre (#PCDATA)>
<!ELEMENT  lugar (#PCDATA)>
<!ELEMENT  estadioprincipal EMPTY>
]>
<ficha>
        <nombre>Roland Garros</nombre>
        <lugar>Paris</lugar>
        <estadioprincipal archivo="estadio.jpeg"/>
</ficha>
```

Consideraciones básicas

Ejemplo:

```
<!ELEMENT  ficha (nombre+, lugar, foto?,
        otrosEstadios*,  emailContacto* | fax*)>
<!ELEMENT  nombre (#PCDATA)>
.....
]>
```

- +: uso obligatorio y múltiple (uno o más)
- *: opcional y múltiple (cero o más)
- ?: opcional pero singular (cero o uno)
- |: or (opciones)
- #PCDATA

Consideraciones básicas (Cont.)

Definiendo Atributos

```
<!ELEMENT  foto EMPTY>
<!ATTLIST  foto
        archivoFuente CDATA  #REQUIRED
        tipo (jpeg | gif )   #IMPLIED
>
```

- REQUIRED: uso obligatorio
- IMPLIED: opcional
- CDATA: acepta cualquier carácter alfanumérico

Consideraciones básicas (Cont.)

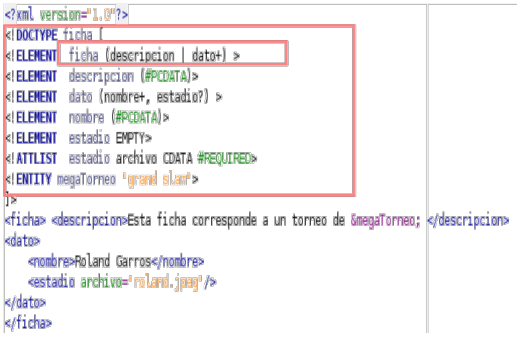
Definiendo Entidades

```
<!ENTITY nombreEntidad "valor">
....
&nombreEntidad;
```

Ejemplo

```
<!ENTITY anio "2014">
<!ENTITY pieDePagina  "Curso Proyecto – &anio;">
```

Un ejemplo más completo



Existen dos tipos de documentos XML

Documentos bien formados:

- Respetan la sintaxis básica
- Un XML mal formado: ver-xml-mal.xml

Documentos válidos:

- Respetan un DTD
- Un XML no válido: ver-xml-invalido.xml
- Usamos un validador-XML

¿Qué pasa si quiero ...?

- Especificar que un elemento dado es de tipo fecha o un número.
- **Asegurarme que un elemento únicamente puede aparecer 3 veces?**
    - Y si quiero indicar que aparezca un mínimo de 3 veces y un máximo de 100?.
- Agregar algún nuevo tipo de elemento o atributo.

Cuando el DTD no alcanza ...

Usamos Schemas

Schemas

- También permiten definir la estructura de un documento XML
- A diferencia de los DTD, utiliza sintaxis XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="ficha">
<xsd:complexType>
        <xsd:sequence>
          <xsd:element name="nombre" type="xsd:string"/>
          <xsd:element name="lugar" type="xsd:string"/>
          <xsd:element name="fechaInicio"  type="xsd:date"/>
        </xsd:sequence>
        <xsd:attribute name="tipo" type="xsd:string" use="required" />
  </xsd:complexType>
......
</xsd:schema>
```

Schemas vs. DTDs

- Los schemas son documentos XML: se pueden procesar como cualquier otro documento XML.
- Los schemas soportan tipos de datos: se pueden definir elementos enteros, de punto flotante, fechas, strings, etc.
- Los schemas son extensibles: se pueden crear nuevos tipos de datos, por ejemplo.
- Los schemas tienen más poder de expresión: se puede especificar, por ejemplo, que cierto valor no tenga más de 2 caracteres.

Ejemplo de Schemas

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="ficha">
<xsd:complexType>
        <xsd:sequence>
          <xsd:element name="nombre" type="xsd:string"/>
          <xsd:element name="lugar" type="xsd:string"/>
          <xsd:element name="fechaInicio" type="xsd:date"/>
          <xsd:element name="estadio" type="xsd:string"
                       minOccurs="2" maxOccurs="9"/>
        </xsd:sequence>
        <xsd:attribute name="tipo" type="xsd:string" use="required" />
  </xsd:complexType>
......
</xsd:schema>
```

Otras notaciones

JSON - YAML

¿De qué estábamos hablando?

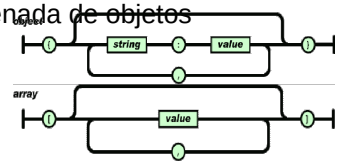
- Intercambio de información entre aplicaciones.
- Definición de datos estructurados.
- Procesamiento en el cliente.
- Procesamiento en el servidor.

JSON – JavaScript Object Notation

- Formato alternativo para el envío y recepción de datos.
- **Es un subconjunto de la notación literal de objetos de Javascript.**
    - Si bien aún no vimos JS, veremos cómo es la notación.
- Se lo conoce también como LJS.
- Es un formato ligero de intercambio de datos.
- Muy popular.

Sintaxis JSON

- **JSON está constituído por dos estructuras:**
    - Objetos: Una colección de pares de nombre/valor
    - Arreglos: Una lista ordenada de objetos



Ejemplo sencillo

```
libro= {
        "titulo": "La casa de los espiritus",
        "precio": "21.90",
        "autor": "Isabel Allende",
        "paginas": "350"
        }

coleccion= [libro1, libro2]
```

Libros en XML

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<Libros>
         <libro>
                <titulo>La casa de los espiritus</titulo>
                <precio>21.90</precio>
                <autor>Isabel Allende</autor>
                <paginas>350</paginas>
         </libro>
         <libro>
                <titulo>El socio </titulo>
                <precio>21.90</precio>
                <autor>John Grisham</autor>
                <paginas>504</paginas>
         </libro>
</Libros>
```

Libros en JSON

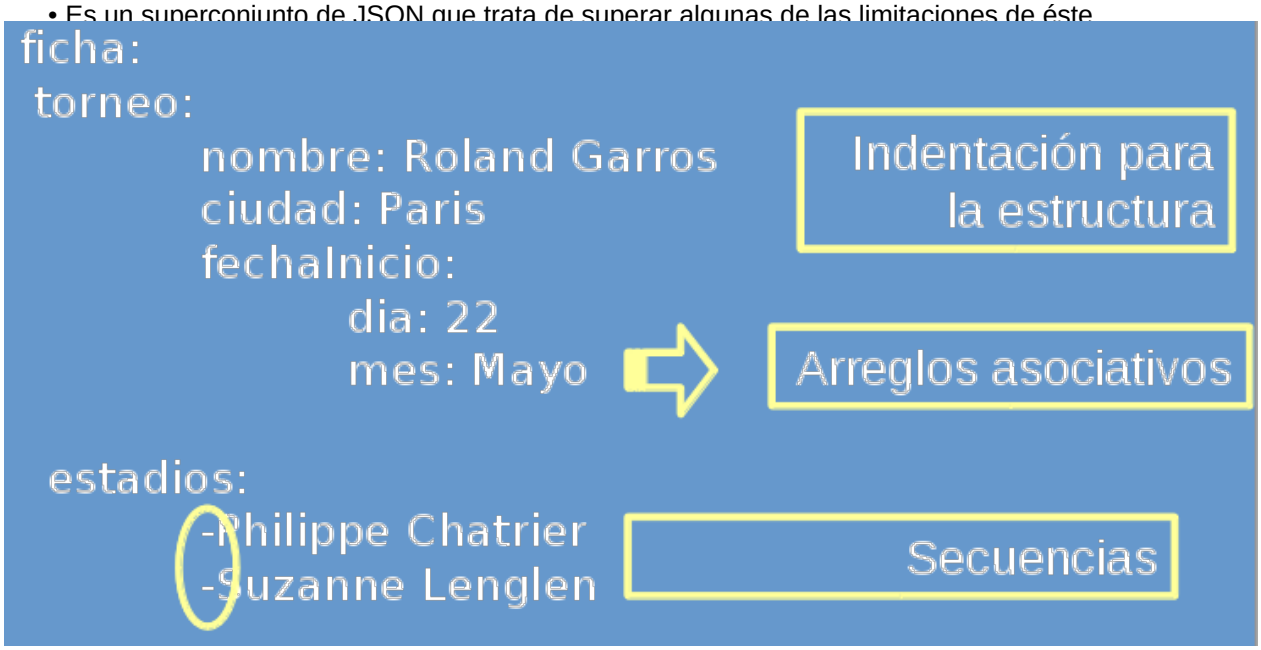
```
"Libros": {
        "libro":[
                {"titulo": "La casa de los espiritus",
                "precio": "21.90",
                "autor": "Isabel Allende",
                "paginas": "350"
                },
                {"titulo": "El socio",
                "precio": "21.90",
                "autor": "John Grisham",
                "paginas": "504"
                },
        ]
}
```

¿Cómo proceso?

- **En el cliente: usando Javascript.**
    - Veremos la clase próxima.
- **En el servidor: usando PHP.**
    - PHP tiene una extensión para JSON: http://php.net/manual/es/book.json.php

YAML – YAML Ain't Markup Language

- Es un superconjunto de JSON que trata de superar algunas de las limitaciones de éste



YAML - Notación resumida

¿Nos suena conocido?

```
ficha:
        torneo:{nombre: Roland Garros, ciudad: Paris,
        fechaInicio: { dia: 22, mes: Mayo },
        estadios: [Philippe Chatrier, Suzanne Lenglen]
```

Ejemplos de usos

databases.yml

```
all:
  propel:
    class:          sfPropelDatabase
    param:
```

schema.yml

```
propel:

###################################
# Tablas - Usuarios del sistema #
###################################

  U_Usuario:
    _attributes:            { phpName: U_Usuario }
    username:               { type: varchar(10), required: true, primaryKey: true }
    clave:                  { type: varchar(70), required: true}
    nombre:                 { type: varchar(20), required: true}
    apellido:               { type: varchar(20), required: true}
    borrado:                { type: boolean, required: true }

# U_Credencial: [ALTA, BAJA, MODIFICACION, CONSULTA, REGULAR, ADMIN]
  U_Credencial:
    _attributes:            { phpName: U_Credencial }
    tipo:                   { type: varchar(12), required: true, primaryKey: true }

  U_Usuario_Credencial:
    _attributes:            { phpName: U_Usuario_Credencial }
    username:               { type: varchar(10), foreignTable: U_Usuario,
foreignReference: username, required: true, primaryKey: true }
    credencial:             { type: varchar(12), foreignTable: U_Credencial,
foreignReference: tipo, required: true, primaryKey: true }

```

Libros en JSON

```
"Libros": {
        "libro":[
                {"titulo": "La casa de los espiritus",
                "precio": "21.90",
                "autor": "Isabel Allende",
                "paginas": "350"
                },
                {"titulo": "El socio",
                "precio": "21.90",
                "autor": "John Grisham",
                "paginas": "504"
                },
        ]
}
```

Libros en YAML

```yaml
libros:
        -titulo   : "La casa de los espiritus"
         precio   : 21.90
         autor    : "Isabel Allende"
         paginas  : 350
        -titulo   : "El socio"
         precio   : 21.90
         autor    : "John Grisham"
         paginas  : 504
```

En resumen ....

- Importancia del intercambio de datos en un formato estándar.
- Procesamiento eficiente.
- Legibilidad para el desarrollador.
- Depende del uso.

---

class: tabla

Algunas comparativas

- Un análisis con algunos números
- Generación de 2.000.000 de usuarios con id y name.
- XML: <user><id>%d</id><name>%s</name></user>
- JSON: {id:%d,name:"%s"}
- Resultados:

	Text	Gzip	Zip duration
XML	91.78M	18.74M	3.38s
JSON	49.78M	17.09M	2.78s
XML overhead	84.38%	9.62%	21.3%

Y entonces... ¿qué usamos?

Como siempre ... depende...

- **XML es más que un formato para describir información**
    - Existen: DTDs y Schemas, XSL, XPath, etc.

- Simplicidad vs. legibilidad vs. performance

Referencias

XML http://www.w3.org/XML/ http://www.w3.org/TR/REC-xml/

Schemas http://www.xml.com/pub/a/2000/11/29/schemas/part1.html http://www.w3.org/XML/Schema.html

JSON http://www.json.org

YAML http://www.yaml.org