INTEGRACION DE SOLUCIONES

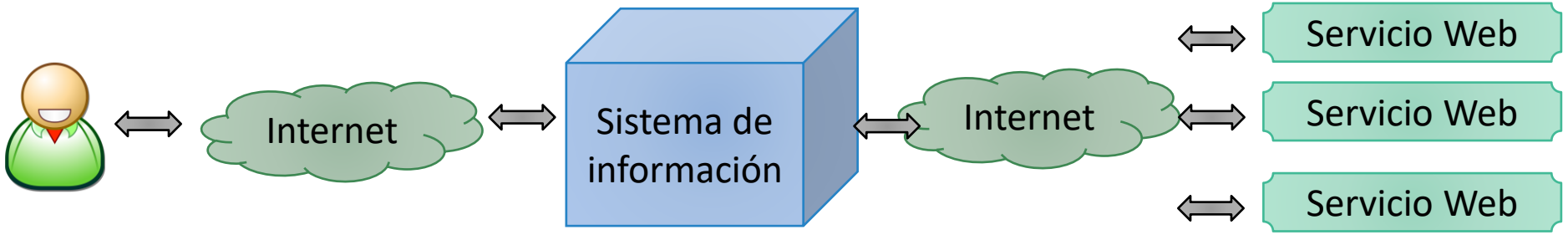
Servicios Web REST

Introducción

¿QUÉ ES UN SERVICIO WEB?

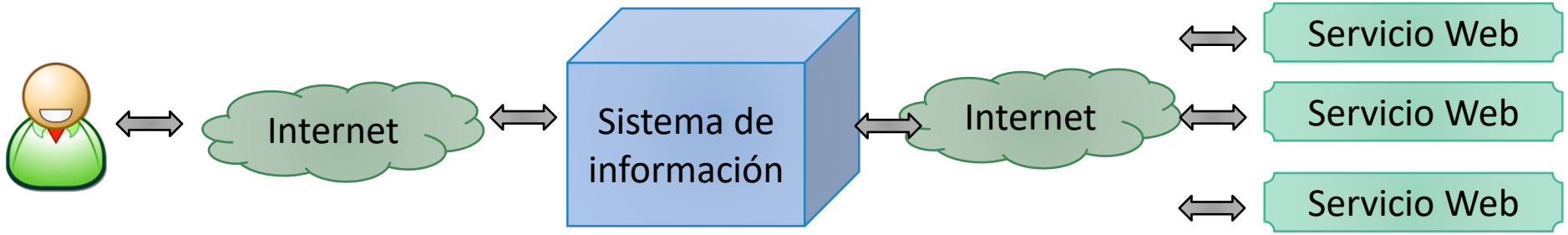
¿Qué es un Servicio Web?

- **El consorcio W3C define los Servicios Web como sistemas software diseñados para soportar una interacción interoperable maquina a maquina sobre una red.**
  - Esta definición de **Servicios Web** alberga muchos tipos diferentes de sistemas, pero los estudiados en el curso se han referido a los **clientes y servidores** que se comunican mediante **mensajes XML** que siguen el estándar **SOAP**.



¿Qué es un Servicio Web?

- Actualmente los **Servicios Web** suelen ser **APIs** (Interfaz de Programación de Aplicaciones) Web que pueden ser accedidas dentro de una red (principalmente Internet), usando un protocolo (HTTP), un formato (XML o JSON) y son ejecutados en el sistema que los aloja (servidor Web).



Estilos de arquitectura de Servicios Web

- En los últimos años los estilos de arquitectura de software han evolucionado de la siguiente manera:
  - **Remote Procedure Calls o RCP**. Orientado a operaciones.
  - **Arquitectura Orientada a Servicios o SOA**. Orientado a mensajes.
  - **Arquitectura Orientada a Recursos o REST (Representational State Transfer)**. Servicios Web implementados a través del protocolo HTTP o protocolos similares, con la restricción de establecer la interfaz a un conjunto conocido de operaciones estándar (GET, POST, PUT, DELETE). Se **centra en interactuar con recursos con estado**, más que con mensajes y operaciones.

¿Por qué otro estilo REST? ¿Problemas con SOAP?

SOAP ofrece una excelente manera de trasferir datos entre aplicaciones, pero:	Junto con los datos, muchos metadatos también necesitan ser transferidos en cada petición y respuesta.
	Esta información extra es necesaria para conocer las capacidades del Servicios Web a consumir.

¿Por qué otro estilo REST? ¿Problemas con SOAP?

SOAP ofrece una excelente manera de trasferir datos entre aplicaciones, pero:	Este intercambio, hace que la carga en la comunicación sea pesada aun para pequeños datos.
	Además, los clientes necesitan crear un proxy para empaquetar y desempaquetar los mensajes SOAP usando WSDL para hacer posible la comunicación.
	El problema con este proxy es que si el servicio es actualizado, pero el cliente no, el consumo del servicio Web no funcionará correctamente.

REST

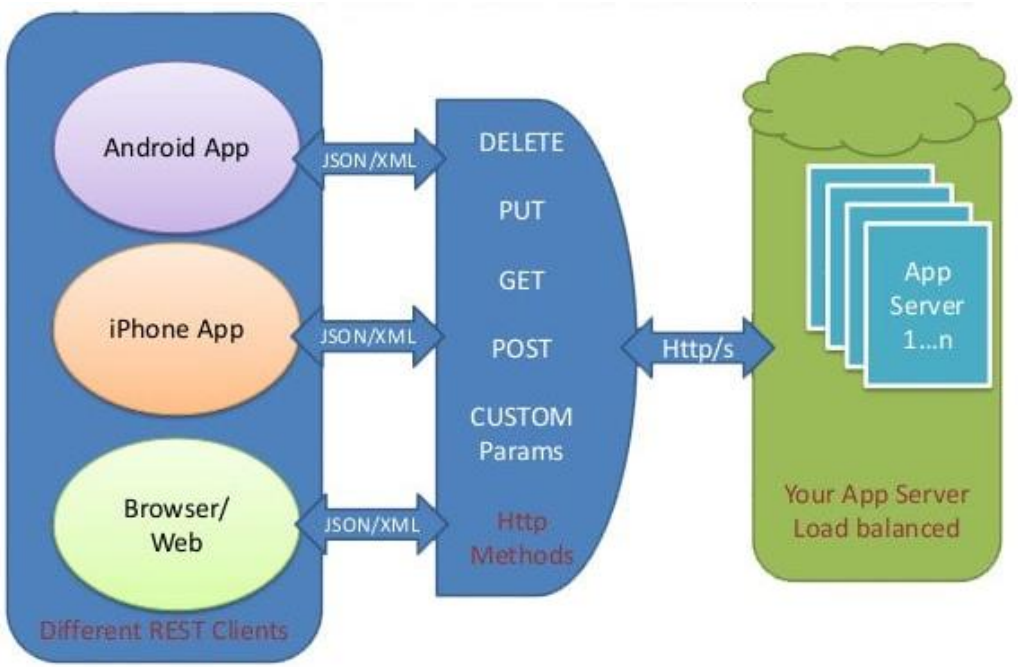
SERVICIOS WEB REST

¿Qué es REST?

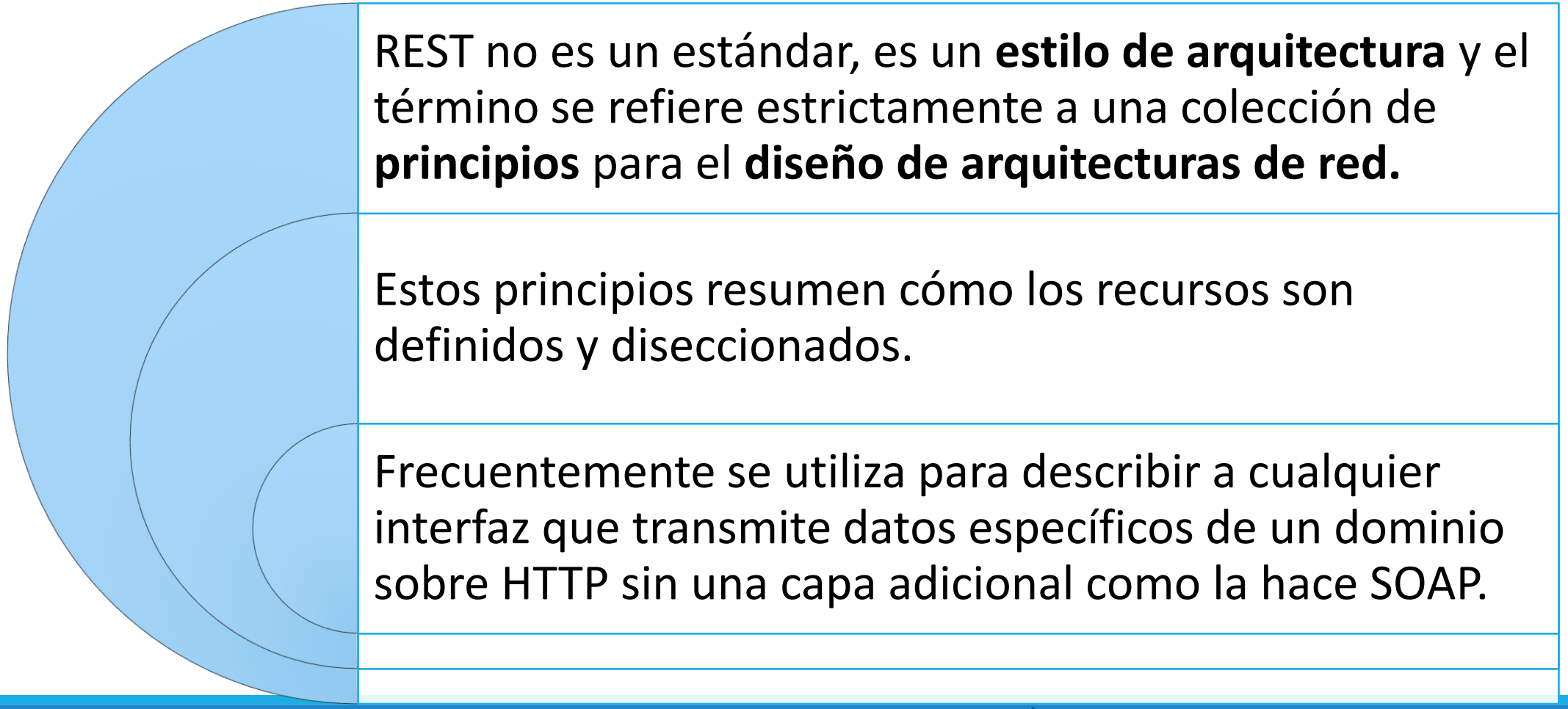
REST (Representational State Transfer) es un **estilo de arquitectura** de software para sistemas distribuidos en la Web.

Se refiere a una interfaz, que consiste en una red de recursos Web relacionada por links y operaciones como GET, DELETE (transiciones de estado).

El término fue introducido en la tesis doctoral de Roy Rielding en 2000, quien es uno de los principales autores de la especificación HTTP.



¿Qué es REST?

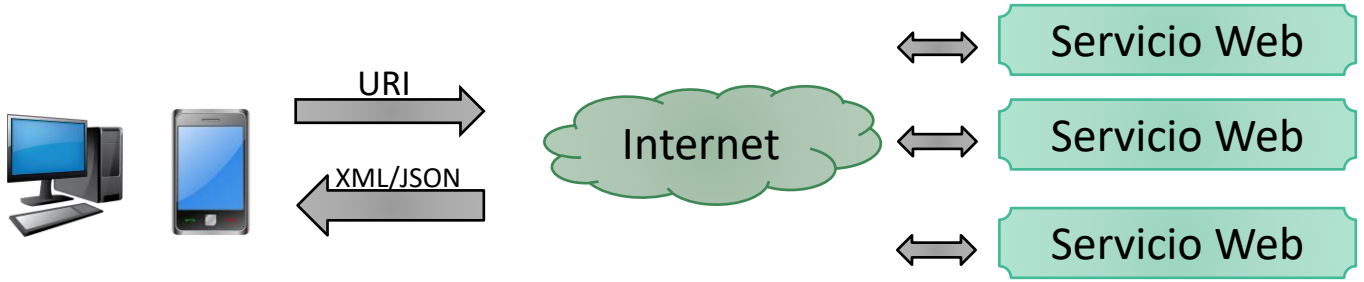

REST no es un estándar, es un **estilo de arquitectura** y el término se refiere estrictamente a una colección de **principios** para el **diseño de arquitecturas de red.**
Estos principios resumen cómo los recursos son definidos y diseccionados.
Frecuentemente se utiliza para describir a cualquier interfaz que transmite datos específicos de un dominio sobre HTTP sin una capa adicional como la hace SOAP.

¿Qué es REST?

- Aunque REST no es un estándar, esta basado en estándares:
  - HTTP - Hypertext Transfer Protocol
  - URL - Uniform Resource Locator
  - Representación de recursos: XML/HTML/JSON
  - Tipos de contenido MIME: text/xml, text/html, text/json

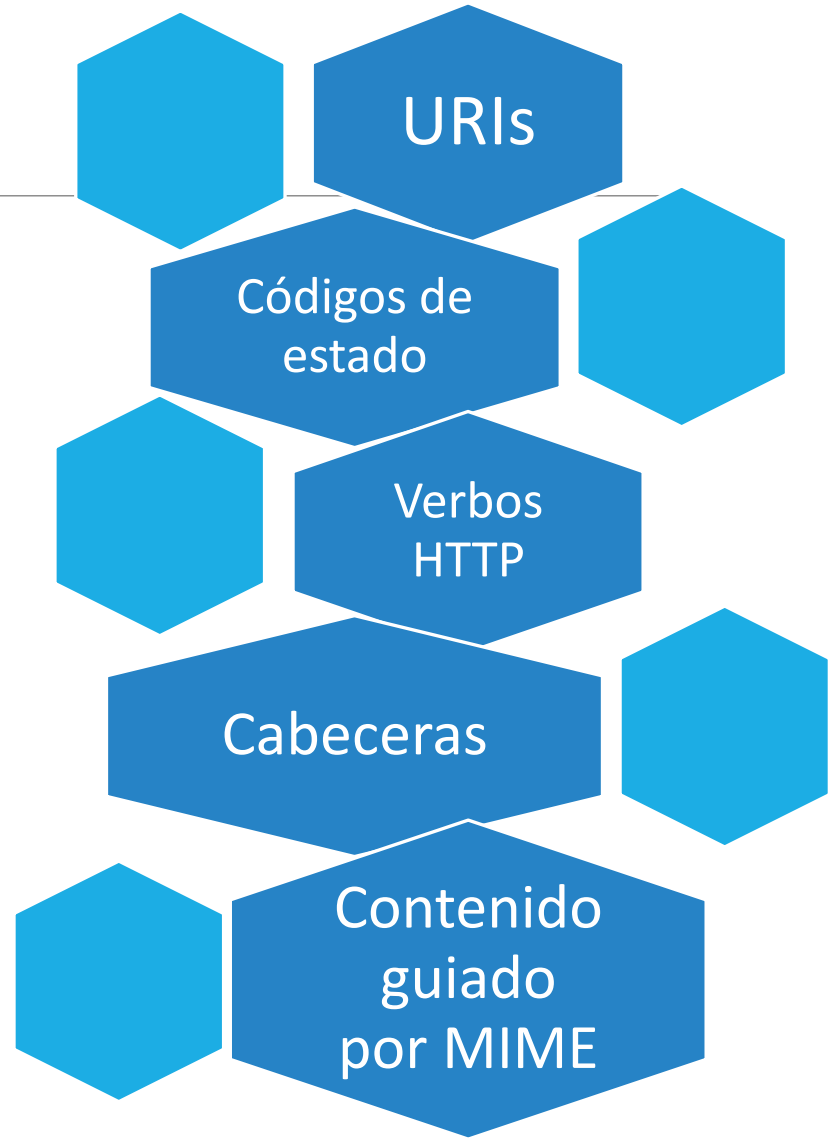
¿Qué es REST?

- Intenta usar todas las características de la Web para el intercambio de mensajes entre computadoras distribuidas.
- Las **URIs identifican los recursos**, los cuales son objetos conceptuales. La representación de tales objetos se distribuye por medio de mensajes a través de la Web.



Elementos participantes en REST

Debido a que la Web evidentemente es un ejemplo clave de diseño basado en REST; consiste del protocolo HTTP con una interfaz uniforme para acceso a los recursos, el cuál consiste de:



URIs

Un identificador de recursos uniforme o URI —del inglés Uniform Resource Identifier— es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

Las “cosas” identificadas por URIs son “recursos”. Aunque es más apropiado decir que los Recursos son identificados mediante URIs.

Operación	URI
Listado	https://www.myserver.com/api/películas/
Detalle	https://www.myserver.com/api/películas/3

Verbos HTTP

- Los verbos más importantes en HTTP son PUT, GET, POST, DELETE.
- Suelen ser comparados con las operaciones asociadas a la tecnología de base de datos: CRUD: CREATE, READ, UPDATE, DELETE.
- Otras analogías pueden también ser hechas como con el concepto de copiar-y-pegar (Copy&Paste).

Acción	HTTP	SQL	UNIX Shell
Create	PUT	INSERT	>
Read	GET	SELECT	<
Update	POST	UPDATE	>>
Delete	DELETE	DELETE	Del/Rm

Verbos HTTP

- Cuando utilizamos REST, HTTP no tiene estado. Cada mensaje contiene toda la información necesaria para comprender la petición. Como resultado, **ni el cliente ni el servidor necesita mantener ningún estado en la comunicación.**

Códigos de estado

- HTTP especifica un conjunto de códigos de estado para cada llamada.
- Es decir, al solicitar un recurso mediante un URI, el servidor regresará un código de estado de la respuesta, por ejemplo: Éxito, Error o No encontrado.

Códigos de estado

Código de estado	Significado
200	Éxito. La respuesta incluirá información pertinente sobre el recurso
201	Recurso creado. La respuesta incluirá la URI del recurso creado.
301	El recurso se ha movido. Debe incluir la URI de la nueva ubicación.
400	Petición errónea. El cliente debe reformular la petición.
401	Sin autorización. Credenciales erróneas para acceder al recurso.
403	Acceso denegado. Se ha autentificado al usuario, pero no tiene permiso de acceder al recuso.
404	Recurso no encontrado.
500	Error de servidor. Algo malo pasó en el servidor. Se debe incluir algún tipo de información.

Cabeceras

- Las Cabeceras HTTP son los parámetros que se envían en una petición o respuesta HTTP, al cliente o al servidor para proporcionar información esencial sobre la transacción en curso.
- Estas cabeceras proporcionan información mediante la sintaxis 'Cabecera: Valor' y son enviadas automáticamente por el navegador o el servidor Web.

Cabeceras

Cabecera de petición	Ejemplo
Accept	Accept: text/plain
Accept-Charset	Accept-Charset: utf-8
Accept-Language	Accept-Language: en-US

Cabecera de respuesta	Ejemplo
Status	Status: 200 OK
Content-Type	Content-Type: text/html; charset=utf-8
Content-Language	Content-Language: en

Contenido guiado por MIME

Se refiere a los encabezados MIME media type ó content type.

Es un identificador del formato del archivo y el contenido que se transmite en Internet.

Esto permite conocer el visor apropiado para el tipo de datos del encabezado enviado.

Se compone de un tipo y un subtipo: tipo/subtipo.

MIME Type
application/javascript
application/json
application/xml
application/zip
text/css
text/html
image/jpeg

¿REST vs los servicios Web estudiados anteriormente

Popularmente se generaliza el concepto de servicio Web con el de servicio Web basado en SOAP. Como hemos visto en apartados anteriores, es posible diseñar servicios Web basados en REST, es decir tomando REST como estilo de diseño.

Por tanto, REST no es una alternativa a los servicios Web, más bien, un servicio Web puede ser implementado mediante SOAP o basado en el estilo REST.

¿Cómo diseñar un Servicio Web basado en REST?

1. Identificar todas las entidades conceptuales que se desean exponer como servicio.
2. Crear una URL para cada recurso. Los recursos deberían ser nombres, no verbos (acciones).

Por ejemplo no utilizar esto:

http://www.service.com/entities/getEntity?id=001

Como podemos observar, getEntity es un verbo.

Mejor utilizar el estilo REST, un nombre: http://www.service.com/entities/001

¿Cómo diseñar un Servicio Web basado en REST?

1. Categorizar los recursos de acuerdo con si los clientes pueden obtener un representación del recurso o si pueden modificarlo.

   Para el primero, debemos hacer los recursos accesibles utilizando un HTTP GET. Para el último, debemos hacer los recursos accesibles mediante HTTP POST, PUT y/o DELETE.

2. Todos los recursos accesibles mediante GET deben devolver la representación del recurso.

¿Cómo diseñar un Servicio Web basado en REST?

3. Ninguna representación debería estar aislada. Es decir, es recomendable poner hipervínculos dentro de la representación de un recurso para permitir a los clientes obtener más información.
4. Especificar el formato de los datos de respuesta mediante un esquema (DTD, W3C Schema, ...).

¿Cómo diseñar un Servicio Web basado en REST?

4. Para los servicios que requieran un POST o un PUT es aconsejable también proporcionar un esquema para especificar el formato de la respuesta.
5. Describir como nuestro servicio ha de ser invocado, mediante un documento WSDL/WADL o simplemente HTML.

Microsoft ASP.NET Web API

IMPLEMENTANDO REST

¿Qué es ASP.NET Web API?

- HTTP no es solo para servir páginas Web.
- También es una plataforma para construir APIs que exponen servicios y datos.
- Casi cualquier plataforma tiene una biblioteca HTTP, por lo que servicios HTTP pueden alcanzar un gran rango de clientes, incluidos navegadores, dispositivos móviles y aplicaciones de escritorio tradicionales.

ASP.NET Web API es un Framework que facilita la creación de servicios RESTful que son capaces de proveer servicios completamente orientados a recursos.

Alternativas de otras compañías

RESTful Web Services in Java
https://jersey.java.net/


```java
package com.example;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

/**
 * Root resource (exposed at "myresource" path)
 */
@Path("myresource")
public class MyResource {

    /**
     * Method handling HTTP GET requests. The returned object will be sent
     * to the client as "text/plain" media type.
     *
     * @return String that will be returned as a text/plain response.
     */
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt() {
        return "Got it!";
    }
}
```

Alternativas de otras compañías

Slim es un micro framework de PHP para desarrollar aplicaciones Web y APIs.

Slim

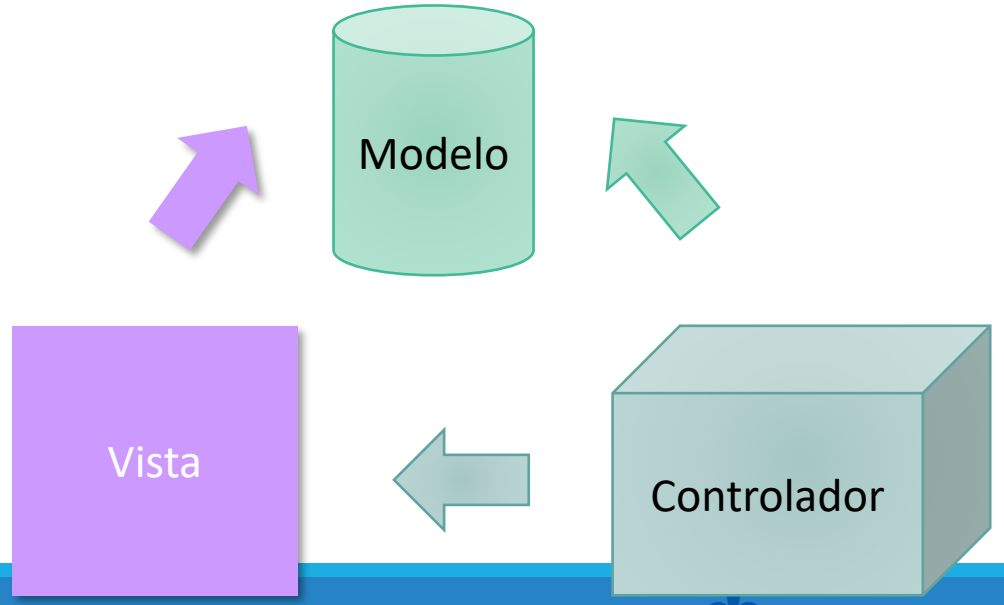
a micro framework for PHP

```php
<?php
use \Psr\Http\Message\ServerRequestInterface as Request;
use \Psr\Http\Message\ResponseInterface as Response;

require 'vendor/autoload.php';

$app = new \Slim\App;
$app->get('/hello/{name}', function (Request $request, Response $response) {
    $name = $request->getAttribute('name');
    $response->getBody()->write("Hello, $name");

    return $response;
});
$app->run();
```

ASP .NET WEB API UTILIZA MVC PARA IMPLEMENTAR SERVICIOS WEB

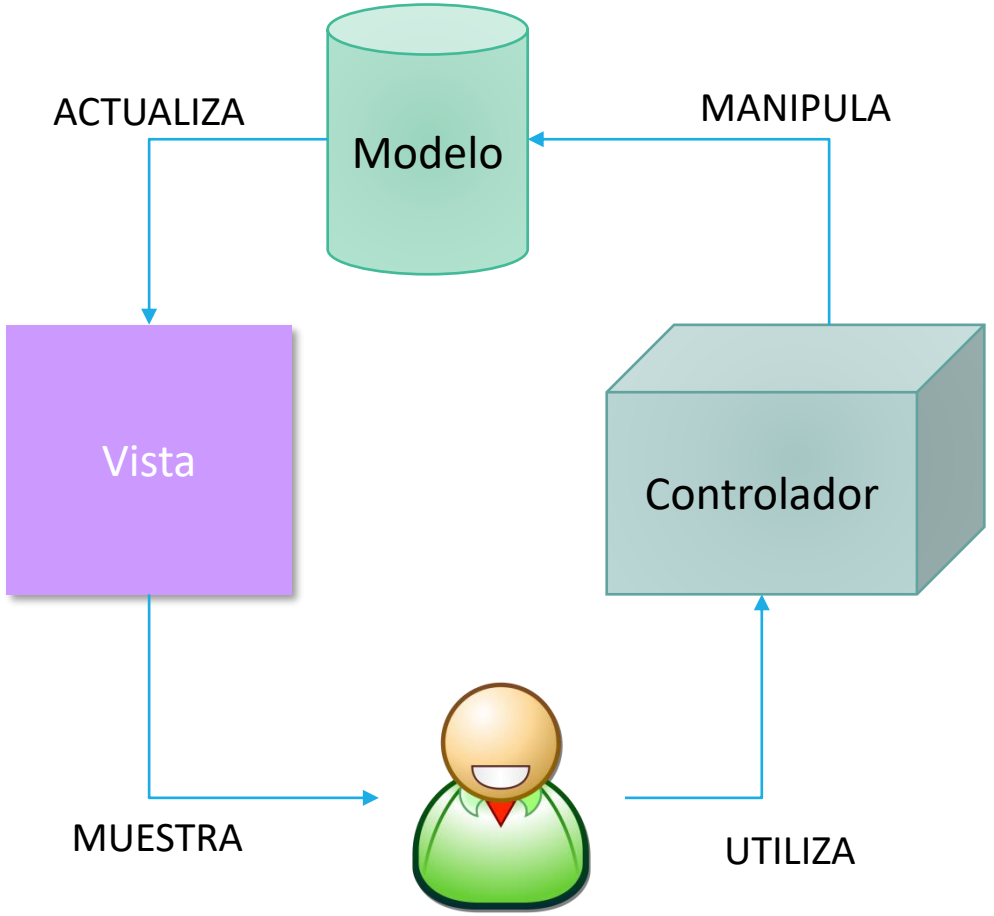
¿Qué es MVC?

- Patrón de arquitectura de software que separa el modelo, la interfaz de usuario y el control de la lógica de una aplicación en tres distintos componentes.
- Presentado por Trygve Reenskaug en 1979 para el Framework Smaltalk (utilizada para crear las interfaces de Apple Lisa y Macintosh).

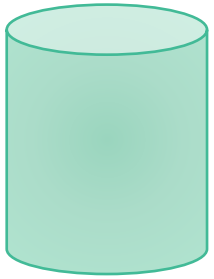


Patrón de MVC

- MVC propone la construcción de tres distintos componentes:
  1. Modelo
  2. Vista
  3. Controlador

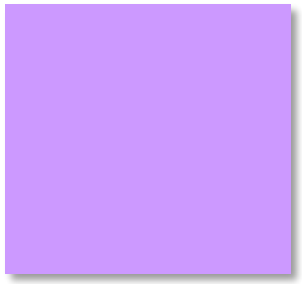


Patrón de MVC



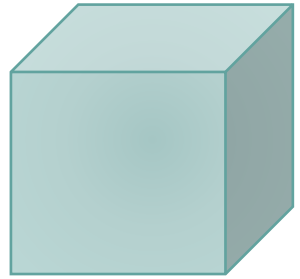
El modelo

Representación de los datos
Lógica del negocio
Obtiene y almacena datos en un almacenamiento persistente (Base de Datos)



La vista

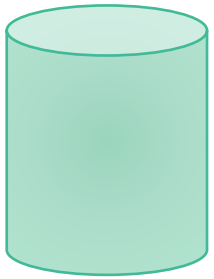
Interfaz de usuario a partir del modelo
Elementos de interacción (HTML, XML)



El controlador

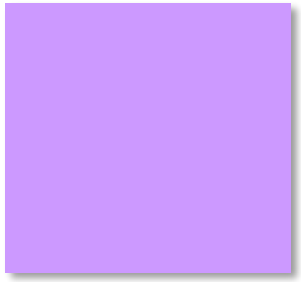
Maneja la interacción con el usuario e invoca cambios al Modelo y Vistas.
Intermediario entre el Modelo y la Vista.

Jerarquía de dependencia en MVC



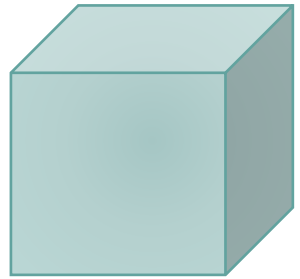
El modelo

El Modelo solo conoce sobre el mismo.
Esto quiere decir, que el código fuente del Modelo no hace referencias ni a la Vista o al Controlador.



La vista

La Vista si conoce al Modelo.
Preguntará al Modelo sobre su estado para saber que mostrar.
De esta manera, la Vista puede desplegar algo que esta basado en lo que el Modelo ha hecho.
La Vista no sabe nada del Controlador.



El controlador

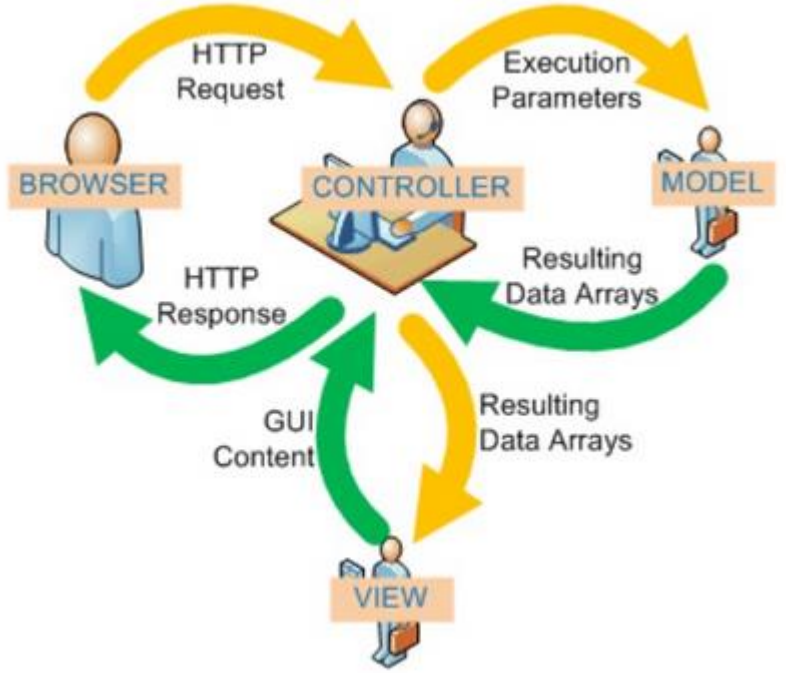
Conoce al Modelo y a la Vista.
Si el usuario realiza una acción, el Controlador sabe que función en el Modelo llamar y también sabe que Vista mostrar al usuario.

Beneficios de MVC

- Fácil de manejar la complejidad
- Desarrollo de aplicaciones más rápido
- Reusabilidad del código
- Desarrollo en paralelo
- Facilita la presentación de información de diferentes maneras donde el código de la aplicación no se ve afectado
- **Ideal para sistemas grandes y distribuidos**
- Gran control sobre el comportamiento de la aplicación

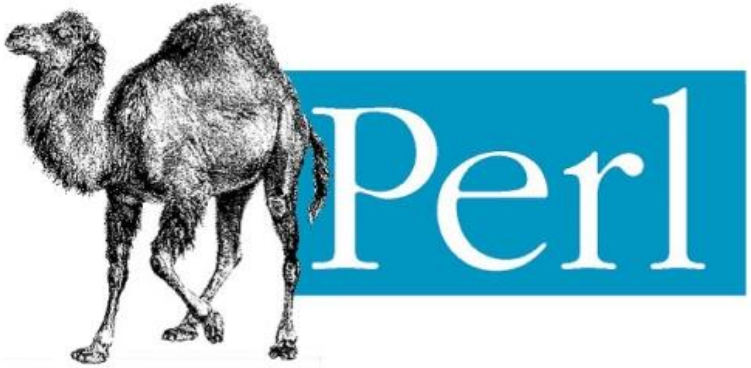
Flujo de MVC

1. El usuario realiza una acción en la interfaz (ej. presiona un botón)
2. El Controlador toma el evento o acción de entrada
3. El Controlador notifica la acción al Modelo, la cuál pudiera involucrar un cambio en el Modelo (ej. edición de datos).
4. Esto genera una nueva Vista que se envía al usuario. La Vista toma los datos del Modelo (ej. lista de usuarios).
5. La interfaz de usuario espera por otra interacción de usuario, lo que inicia un nuevo ciclo.



¿Donde podemos usarlo?

Prácticamente esta disponible en toda clase de sistemas y tecnologías (Java, Ruby, Python, Perl, PHP, Flex, Net, etc.)



Actividad

- Elabora un mapa conceptual, infografía u organizador gráfico, cuadro sinóptico del contenido recién abordado en esta presentación.

Gracias por su atención

Referencias

ASP.NET MVC 4 and the Web API. (diciembre de 2012). Jaime Kurtz. aPress. Obtenido de: http://sd.blackball.lv/library/ASP.NET_MVC_4_and_the_Web_API.pdf

REST Vs Web Services (2006). Rafael Navarro. Modelado, Diseño e Implementación de Servicios Web. Obtenido de: http://users.dsic.upv.es/~rnavarro/NewWeb/docs/RestVsWebServices.pdf

RESTful Web Services(2007). Leonard Richarson & Sam Ruby. O'Really. Obtenido de: https://www.crummy.com/writing/RESTful-Web-Services/RESTful_Web_Services.pdf

Getting Started with ASP.NET Web API 2. (2015). Mike Watson. Microsoft Docs. Obtenido de: https://docs.microsoft.com/en-us/aspnet/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api