



Connecting JSON, REST and AngularJS

Introducción

Objetivo

- Orientación en REST, JSON y AngularJS
- ¿Cómo usar estas tecnologías de forma conjunta?
- Explorar los conceptos con ejemplos

Escenario

- Caché 2016.2 + Atelier





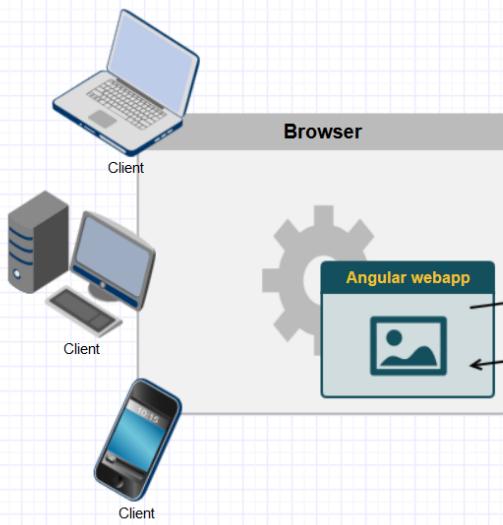
Connecting JSON, REST and AngularJS

Trabajando con JSON

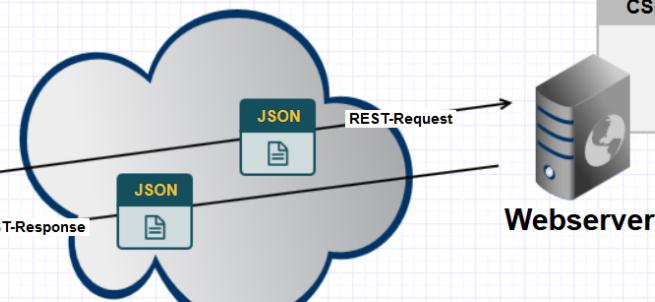


Esquema general

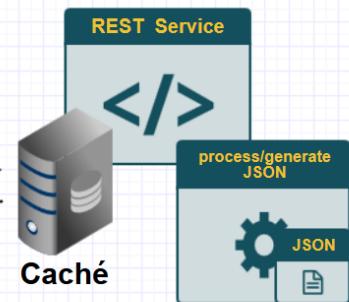
Client-Side



Internet / http(s)

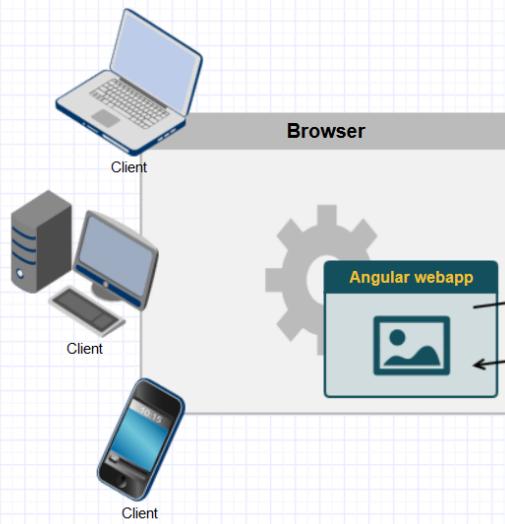


Server-Side

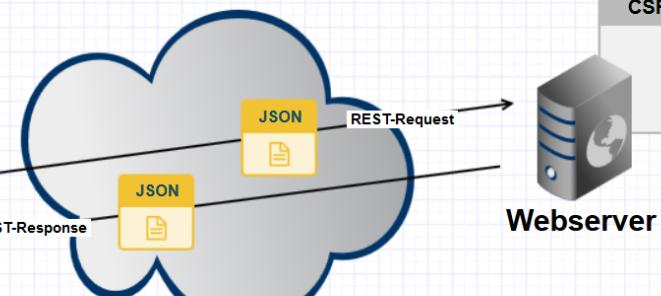


Uso de formato JSON

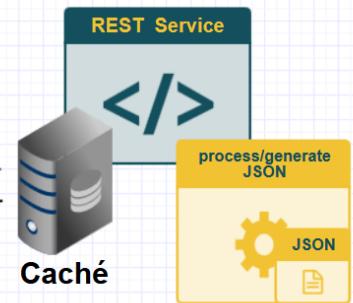
Client-Side



Internet / http(s)



Server-Side



JSON – Introducción

- <http://json.org>
- JSON: **JavaScript Object Notation**
- Formato de lectura legible
- Contiene datos **sin esquema**
- Basado en parejas clave/valor
- Claves y cadenas literales se representan entre comillas dobles
- Se ignoran los espacios en blanco fuera de los nombres de claves y valores



JSON – Representación de tipos de datos

Tipo	Ejemplo
string	"iPhone6"
number	530.6
true	true
false	false
null – no value	null
object	{"model" : "iPhone5"}
array	["red", "green", "blue"]



JSON – Objetos

Los objetos se encierran entre llaves — { ... }

Ejemplo...

```
{ "model" : "iphone5" }
```

... es un objeto que contiene una propiedad “model” con el valor “iphone5”



JSON – Objetos

En caso de varias parejas clave-valor, estas se separan por comas

Ejemplo...

```
{  
    "model" : "iphone5",  
    "color" : "red",  
    "manufacturer" : "Apple"  
}
```



JSON – Arrays

- Los Arrays se encierran entre corchetes — [...]
- Listas ordenadas que contienen cualquier tipo de datos
- Cada elemento se separa por coma
- Ejemplo...

```
[ "red" , "green" , "blue" ]
```

```
[ { "model" : "iphone4"} ,  
  { "model" : "iphone5"} ]
```



JSON – Si has prestado atención

Seras capaz de resolver...

¿Cuáles de estos cadenas es una sintaxis JSON válida?

- A { model } : { iphone5 }
- B { "model" : "iphone5" }



JSON – Si has prestado atención

Seras capaz de resolver...

¿Cuáles de estos cadenas es una sintaxis JSON válida?

Incorrecto

```
{ model } : { iphone5 }
```

Correcto

```
{ "model" : "iphone5" }
```



JSON – Si has prestado atención

Seras capaz de resolver...

¿Qué es necesario cambiar para que sea un Array JSON válido?

```
[ "red" , "green" , blue ]
```



JSON – Si has prestado atención

Seras capaz de resolver...

¿Qué es necesario cambiar para que sea un Array JSON válido?

Incorrect

```
[ "red" , "green" , blue ]
```

Correcto

```
[ "red" , "green" , "blue" ]
```



JSON – Si has prestado atención

Seras capaz de resolver...

¿Cómo corregir la cadena para hacer un formato JSON válido?

```
[ { "model" : "iphone5" } , ]
```



JSON – Si has prestado atención

Seras capaz de resolver...

¿Cómo corregir la cadena para hacer un formato JSON válido?

```
[ { "model" : "iphone5" } , ]
```

Un unico objeto

```
[ { "model" : "iphone5" } ]
```

Añadir otro objeto

```
[ { "model" : "iphone5" } ,  
 { "model" : "nexus6" } ]
```



JSON – Si has prestado atención

Seras capaz de resolver...

¿Cómo corregir la cadena para hacer un formato JSON válido?

```
[ ["model" : "iphone5"] ]
```



JSON – Si has prestado atención

Seras capaz de resolver...

¿Cómo corregir la cadena para hacer un formato JSON válido?

Incorrecto

```
[ ["model" : "iphone5"] ]
```

Correcto

```
[ { "model" : "iphone5"} ]
```



JSON vs. XML

JSON

```
[  
  { "model" : "iphone5" } ,  
  { "model" : "iphone6" }  
]
```

- Ligero
- Sin esquema
- Estructuras de datos complejas creadas manualmente
- Integración con Javascript

XML

```
<devices>  
  <device model="iphone5"></device>  
  <device model="iphone6"></device>  
</devices>
```

- Muy pesado
- Estricto al esquema
- Nuevos tipos de datos completos reusables en el esquema
- Validable contra el esquema



JSON en Caché

- API 2016.2
- Implementada en el kernel de Caché (muy rápida)
- Creación de objetos JSON — **%DynamicObject**

```
Set json = {"model": "iphone5"}
```



%DynamicObject API

- Se pueden establecer propiedades directamente

```
set phone = {"model": "iphone5"}  
set phone.price = 115.50
```

- Podemos obtener el valor de una propiedad por su nombre

```
write phone.%Get("price")  
write phone.%IsDefined("manufacturer")
```

- Podemos establecer el valor o eliminarlo a través del nombre

```
do phone.%Set("manufacturer", "Apple")  
do phone.%Remove("price")
```



%DynamicObject API

- Convertir de %DynamicObject a JSON string

```
set phone = {"model": "iphone5"}  
write phone.%ToJson()
```

- Convertir a %DynamicObject desde un JSON string

```
set jsonString = '{"model": "'iphone5'"}'  
set obj = ##class(%DynamicObject)  
    .%FromJSON(jsonString)
```



%DynamicObject API

- %GetIterator() devuelve un iterador util para recorrer todas sus propiedades
- Ejemplo...

```
set iterator = phone.%GetIterator()  
while iterator.%GetNext(.key,.value) {  
    write !, "Phone property name: ",key  
    write !, "Phone property value: ",value  
}
```



%DynamicArray API

- Creación de JSON arrays — **%DynamicArray**

```
set array = ["iphone5", "nexus6"]
```

- El indice es 0-based (y no 1-based como \$list, por ejemplo)
- Obtener el número de elementos del Array:

```
write array.%Size()
```

- Valor del elemento i-esimo:

```
write array.%Get(0)
```

```
write array."1"
```



%DynamicArray API

- Establecer valores en el array:

```
set array."1" = "HTC One"  
do array.%Set(0,"LG G4")
```

- Establecer un indice mayor que el tamaño insertará nulos en los espacios intermedios

Ejemplo...

```
USER>set array = [1,2,3]  
USER>set array."10" = 10  
USER>write array.%TOJSON()  
[1,2,3,null,null,null,null,null,null,null,10]
```



%DynamicArray API

- %Push() añade elemento al final del array
- %Pop() saca el elemento del final del array y lo devuelve
- **%DynamicArray** tambien implementa los métodos
 - %GetIterator()
 - %ToJSON()
 - %FromJSON()
 - %IsDefined() – El parámetro no será una propiedad sino un índice



A dark blue background featuring a complex network graph with numerous small white dots representing nodes and connecting lines representing edges, creating a sense of data connectivity and complexity.

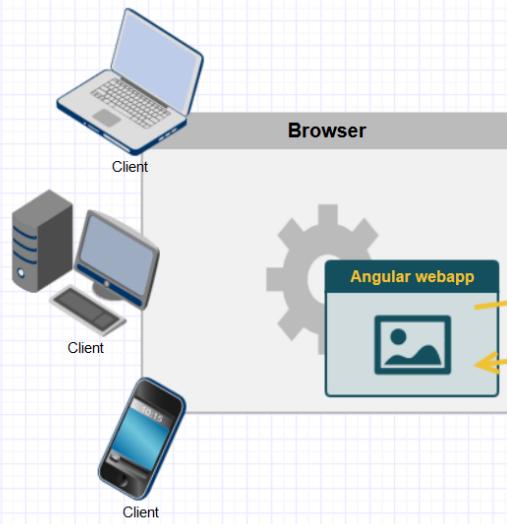
Connecting JSON, REST and AngularJS

Trabajando con REST

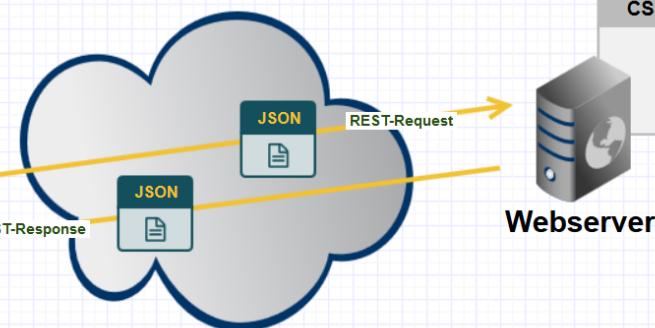


REST – Introducción

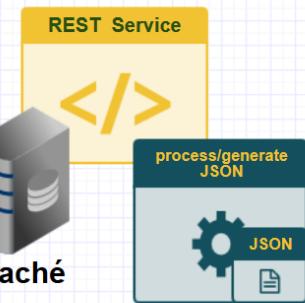
Client-Side



Internet / http(s)



Server-Side



REST – Introducción

- REST: **Representational State Transfer**
- Es un patrón de diseño para servicios basados en web
- Diseño **RESTful**:
 - Cliente-Servidor
 - Sin estado
 - Interfaz uniforme
 - Protocolo: HTTP
 - Formato de intercambio: JSON, XML, etc.
 - Sintaxis: URLs



REST – un poco de HTTP

- HTTP: HyperText Transfer Protocol
- REST se implementa sobre HTTP
- Usa "verbos" para discriminar entre los tipos de solicitud
- Con REST, los verbos se mapean con funciones "CRUD" para aplicaciones con bases de datos

Verbo	Operación CRUD
POST	Create
GET	Read
PUT	Update
DELETE	Delete

REST – URLs

- En REST las URLs contienen información sobre:
 - Protocolo esperado
 - Localización del servidor
 - Recurso solicitado

http://www.mobilephones.com/phones/iPhone5



Protocolo

Dirección del Servidor

Recurso solicitado

REST – Diseño de URLs RESTful

- Es una URL RESTful?

<http://www.example.com/phonestore/getorder?id=12345>

- No, aunque funcione como se espera y devuelva datos.
 - El prefijo "get" no es necesario en la URL; El verbo HTTP ya lo deja claro
- Un mejor ejemplo:

<http://www.example.com/phonestore/orders/12345>
- Un punto importante acerca de la arquitectura REST: El criterio RESTful no se ciñe a restricciones técnicas.



REST – Usando REST en Caché

- Para publicar servicios REST en Caché, se hace uso de una aplicación Web
- Las solicitudes se enrutan hacia la **Dispatch Class** definida para su procesamiento

The screenshot shows the 'Application Roles' tab of the Caché configuration interface. The 'Name' field is highlighted with a red box and contains the value '/phoneapp/api'. Below it, a note says 'Required. (e.g. /csp/appname)'. A large white arrow points downwards from this field to the 'Dispatch Class' field. The 'Dispatch Class' field is also highlighted with a red box and contains the value 'demo.phones.restserver'. The 'CSP File Settings' section is partially visible below.



REST – Código de Dispatch Class

- Las clases Dispatch heredan de **%CSP.REST**
- Las clases REST contienen un bloque XData llamado **UrlMap**
- El tag **<Routes>** contiene un número entradas **<Route>** para cada patrón de URL que puede ser manejada por el servicio.

```
XData UrlMap [ XMLNamespace = "http://www.intersystems.com/urlmap" ]
{
  <Routes>
    <Route Url="/phones" Method="GET" Call=" GetAllPhones" />
    <Route Url="/phones/:id" Method="GET" Call="GetPhone" />
  </Routes>
}
```



REST – Enrutamiento UrlMap

```
<Route Url="/phones" Method="GET" Call="GetAllPhones" />
```

- Cada **<Route>** contiene los atributos **Url**, **Method**, y **Call**.
- **Url** especifica el patrón sobre el que se casa la URL invocada
 - No incluye el prefijo de la aplicación CSP
- **Method** es el verbo HTTP
- **Call** apunta al **ClassMethod** en la clase REST, o en otra clase.

```
<Route Url="/phones/:id" Method="GET" Call="GetPhone" />
```

- Se especifican parámetros anteponiendo ":" que se pasan como argumentos al método invocado



REST – Procesando una Solicitud

GET http://localhost:57779/phoneapp/api/**phones**/5

XData UrlMap [XMLNamespace = "http://www.intersystems.com/urlmap"]

```
{  
  <Routes>  
    <Route Url="/phones" Method="GET" Call="GetAllPhones" />  
    <Route Url="/phones/:id" Method="GET" Call="GetPhone" />  
  </Routes>  
}
```

GetPhone(**id** As %String = "") As %Status

```
{ ... }
```



REST – Procesando una solicitud

- Escribir resultados sobre el Device actual (CSP == %response)
- Devolver el status

```
GetPhone(id As %String = "") As %Status
{
    set phoneObject = ##class(demo.phones.phone).%OpenId(id)
    write phoneObjectToJsonString()
}
quit 1
```



REST – Resumen

- **Detalles de implementación en Caché:**
 1. Todas las solicitudes se manejan por una aplcación Web definida
 2. La aplicación CSP application dirige la solicitud hacia la Dispatch Class
 3. El UrlMap de la Dispatch Class vincula el patrón de la URL con un método de clase



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route  
Url="http://localhost:57772/phoneapp/phones/"  
Method="GET"  
Call="GetPhones" />
```



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route  
Url="http://localhost:57772/phoneapp/phones/"  
Method="GET"  
Call="GetPhones" />
```

No. La Url no puede contener la ruta absoluta.



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route Url="/phones"  
       Call="GetPhone" />
```



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route Url="/phones"  
       Call="GetPhone" />
```

No. The Method attribute is not set.



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route Url="/phones/:id"  
       Method="GetPhone"  
       Call="GET" />
```



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route Url="/phones/:id"  
       Method="GetPhone"  
       Call="GET" />
```

No. The Method and Call attributes are reversed.



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route Url="/phones"  
       Method="GET"  
       Call="demo.phones:GetPhones" />
```



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route Url="/phones"  
       Method="GET"  
       Call="demo.phones:GetPhones" />
```

*Sí! De esta manera se indica que el método está en otra clase
"<class>:<method>"*



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route Url="/phones/:model/:manufacturer"  
       Method="GET"  
       Call="GetPhoneTypes" />
```



REST – Si has estado atento

Seras capaz de resolver...

¿Es esta una definición de ruta válida?

```
<Route Url="/phones/:model/:manufacturer"  
       Method="GET"  
       Call="GetPhoneTypes" />
```

Sí



A dark blue background featuring a complex network graph with numerous small white dots representing nodes and connecting lines representing edges, creating a sense of data connectivity and complexity.

Connecting JSON, REST and AngularJS

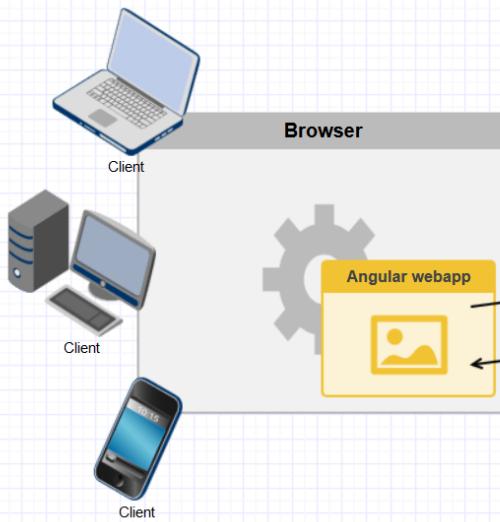
Trabajando con AngularJS



AngularJS — Introduction

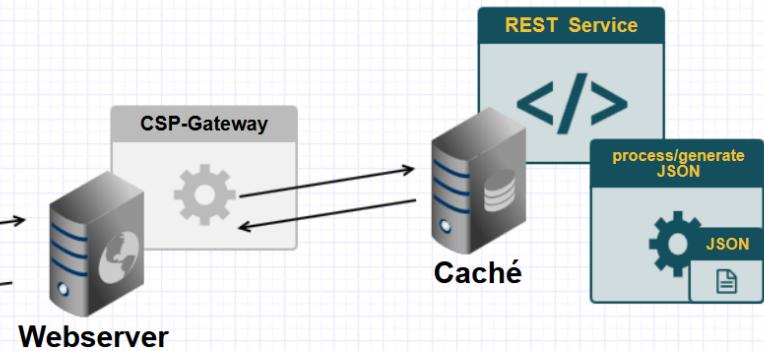
Angular.JS role in overall system

Client-Side



Server-Side

Internet / http(s)



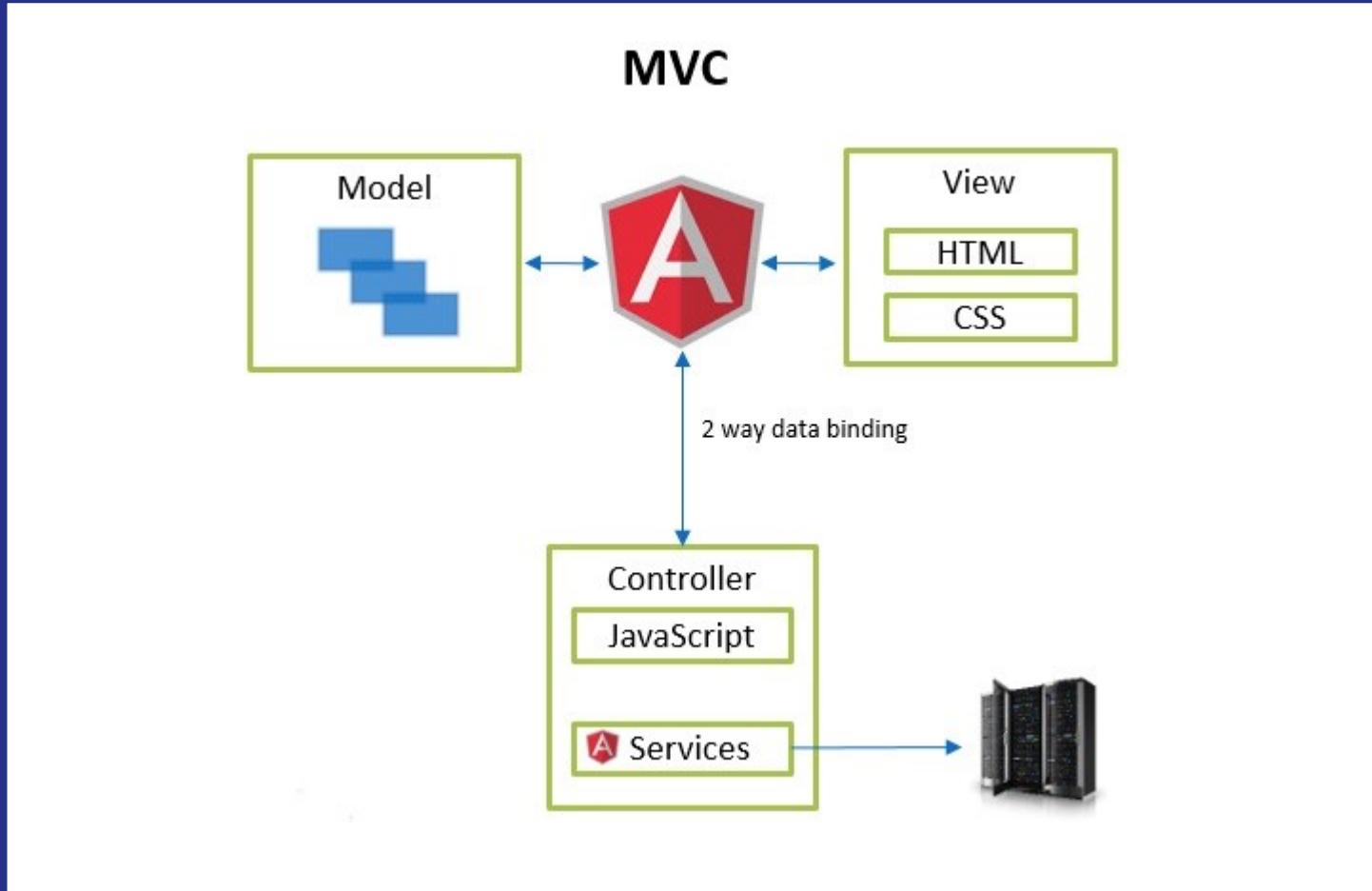
AngularJS — Patrón de diseño MVC

- **Model — View — Controller**
- Patrón de diseño para construir aplicaciones
 - Separación de responsabilidades
 - modulariza el código de la aplicación
- Ejemplo:

Model	Database server (Caché)
View	Template (HTML)
Controller	Client code (JavaScript)



AngularJS MVC



AngularJS — Controllers y Views

- **Angular.js** es un framework JavaScript para construir aplicaciones Web
- Controllers
 - Funciones JavaScript
 - Gestionar entradas del usuario
 - Realizar solicitudes a las APIs (nuestro servicio REST)
- View
 - Página HTML
 - Utiliza una sintaxis especial para conectar las entradas al Controller
- Se establecen **Data bindings** entre Controllers y Views



AngularJS — Data Binding

- Por convenio, el modulo **\$scope** se usa para asociar datos entre los **Controllers** y **Views**.
- En el código del Controller, se utiliza la sintaxis tradicional para establecer propiedades
 - `$scope.property = data;`
- En la plantilla HTML:
 - La sintaxis con doble llave asocia el valor para un vinculo "one-way"
 - Ejemplo: `<h1>{{name}}</h1>` para mostrar el valor de **\$scope.name**
 - El atributo **ng-bind** es equivalente a la sintaxis con doble llave
 - Ejemplo: `<h1 ng-bind="name">` para mostrar el valor de **\$scope.name**
 - El atributo **ng-model** se usa para establecer un vinculo de datos **two-way**
 - Ejemplo: `<input type="text" ng-model="name">`
 - Muestra el valor actual de **\$scope.name**
 - Adicionalmente el texto tecleado se establecerá en **\$scope.name**



AngularJS — modulo \$http

- Angular proporciona un módulo **\$http** para realizar solicitudes HTTP
- El modulo tiene métodos para cada verbo HTTP:
 - Ejemplo: `$http.get()`, `$http.post()`, `$http.put()`, `$http.delete()`
- Las solicitudes HTTP se ejecutan de manera asíncrona
 - Se utilizan métodos de Callback para manejar la respuesta con éxito o con error



Gracias!

