



PREPÁRATE  
PARA SER EL  
**MEJOR**



+ **ENTREMIENTO  
EXPERIENCIA**



**BIENVENIDOS.**



# SERVICIOS REST CON NET CORE (BACK-END)

Sesión 04

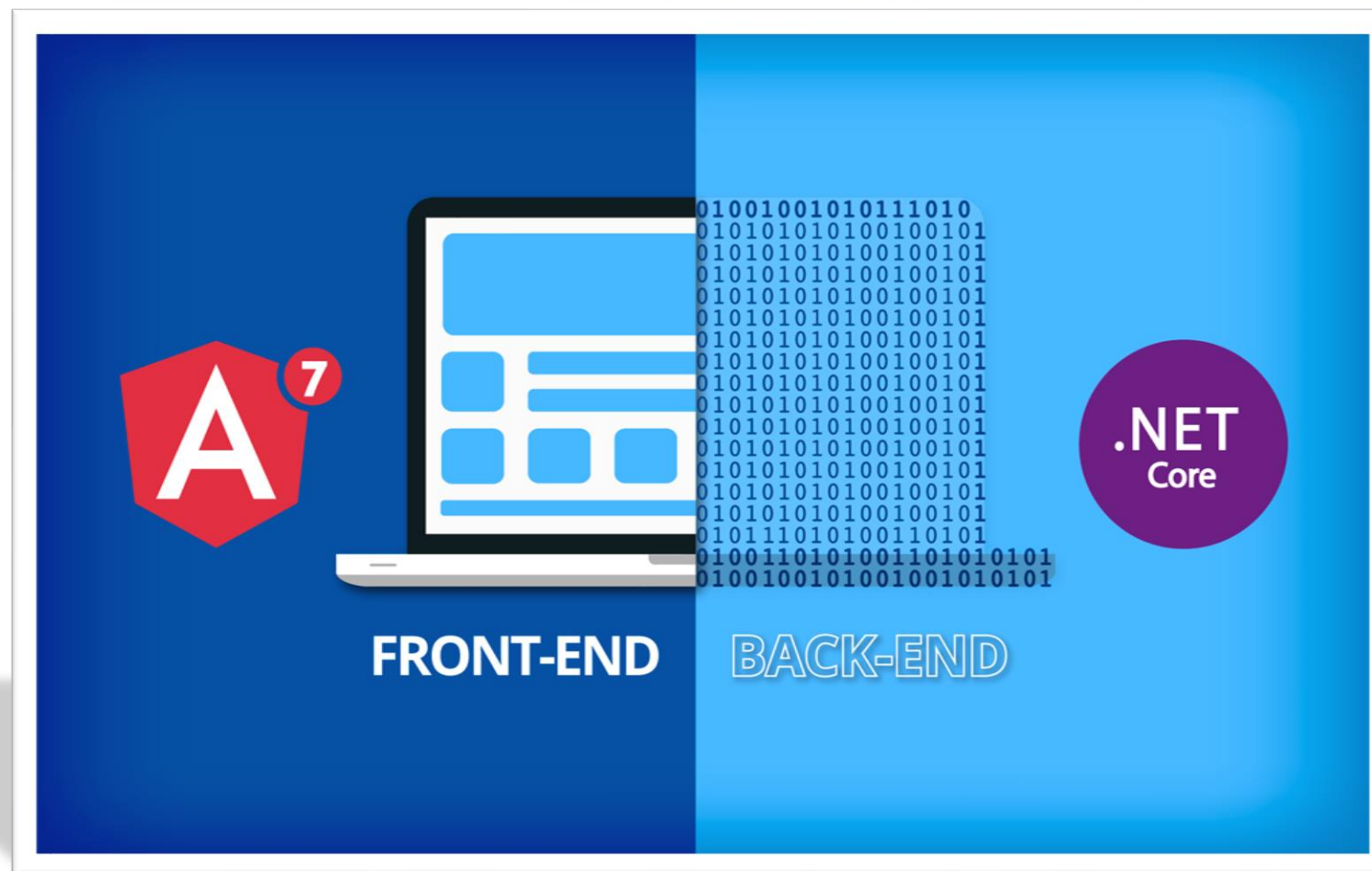
**Ing. Erick Arostegui Cunza**  
**Instructor**

[earostegui@galaxy.edu.pe](mailto:earostegui@galaxy.edu.pe)





# Desarrollo Full-Stack





## AGENDA

### Implementando Web API

- ▶ Introducción a Web API.
- ▶ Creación del Web API.
- ▶ Exponiendo métodos CRUD de la capa de servicios
- ▶ Uso de rutas.
- ▶ Gestión de errores y excepciones.



# REST es...

# Representational State Transfer



**REST tiene la intención de evocar una imagen de cómo se comporta una aplicación web bien diseñada; es decir; una red de páginas web (una máquina de estado virtual) ...**

**... donde el usuario avanza a través de la aplicación mediante enlaces seleccionados (estados de transiciones) ...**

**... dando como resultado que la página siguiente (que representa el siguiente estado de la aplicación) comience a transferirse al usuario y se represente para su uso**

**Roy Fielding**

Architectural Styles and the Design of Network-based Software Architectures

<http://bit.ly/1rbtZik>





## Introducción a REST

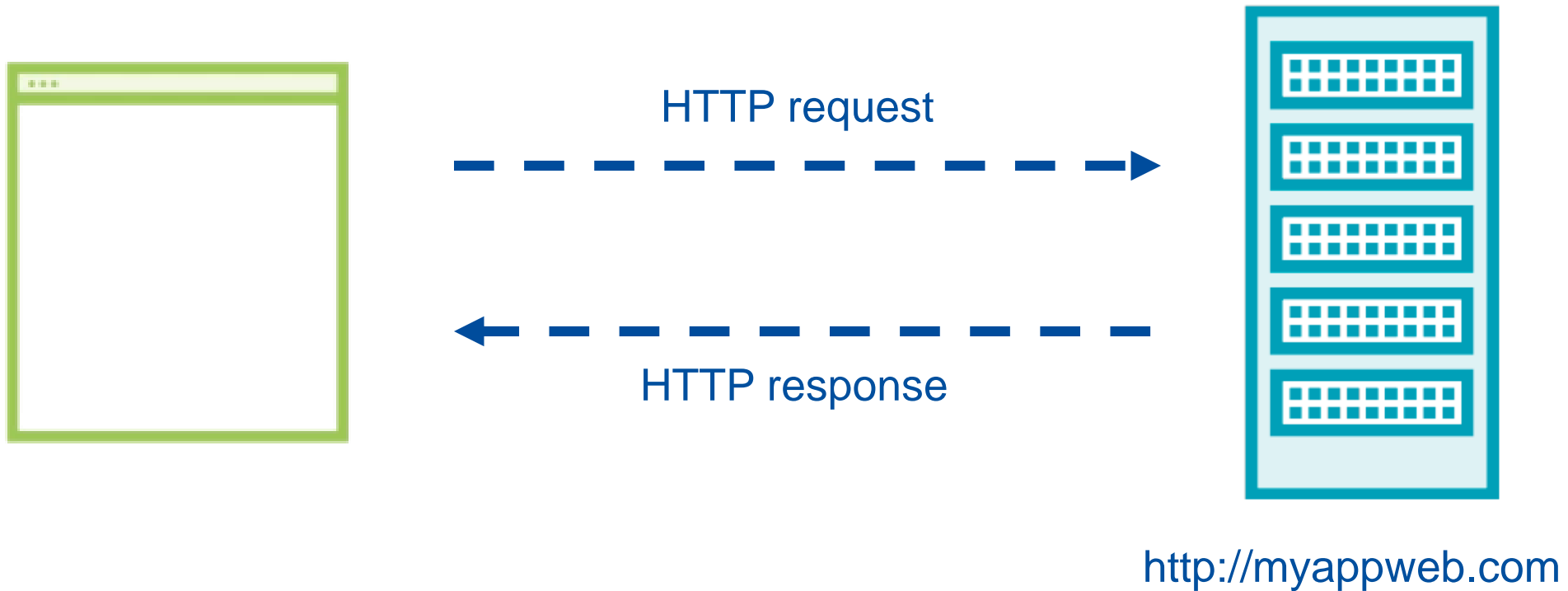


**REST es un estilo arquitectónico, no un estándar, usamos estándares para implementarlo.**

**REST es un protocolo agnóstico**



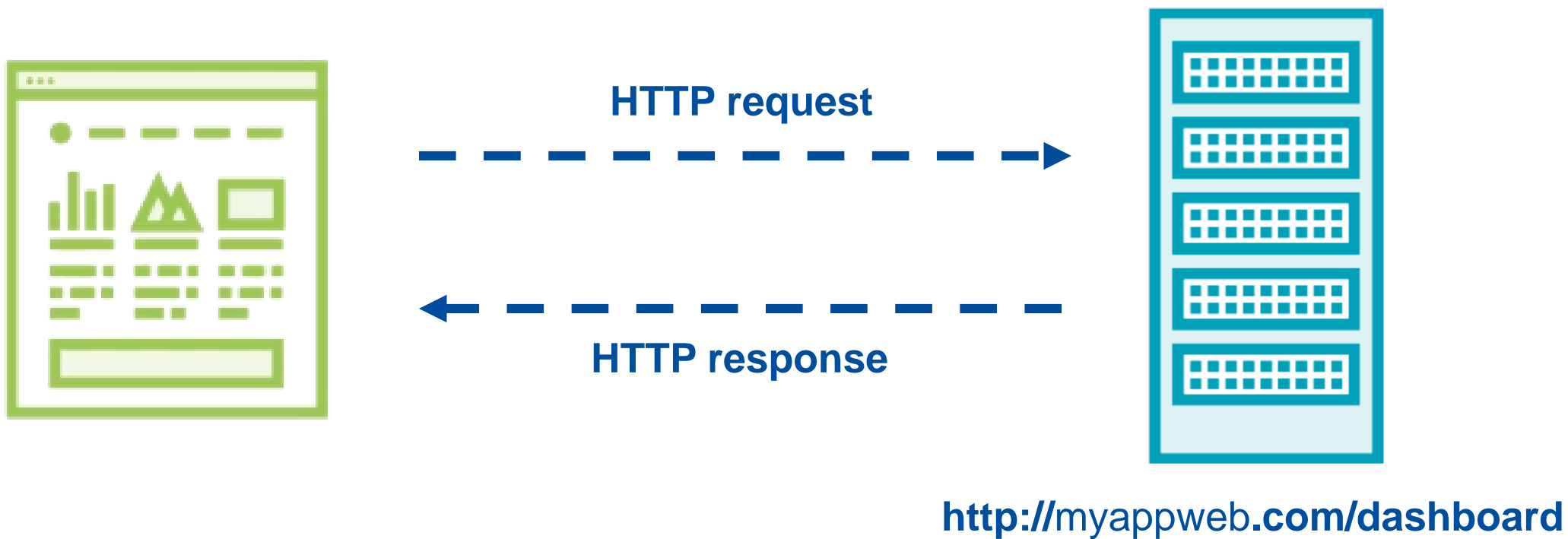
## Representational State Transfer







## Representational State Transfer





## Aprender de qué se tratan las restricciones REST



REST es normado por 6 restricciones (una opcional)

Una restricción es una decisión de diseño que puede tener un impacto positivo y negativo



## Aprender de qué se tratan las restricciones REST

### Cliente-Servidor

El cliente y servidor  
están separados

(cliente y servidor  
pueden evolucionar  
por separado)

### Statelessness

El estado está  
contenido dentro de  
la solicitud

### Cacheable

Cada mensaje de  
respuesta debe  
indicar  
explícitamente si se  
puede almacenar en  
caché o no.



## Aprender de qué se tratan las restricciones REST

### Sistema en capas

El cliente no puede determinar a que capa esta conectada

### código bajo demanda (opcional)

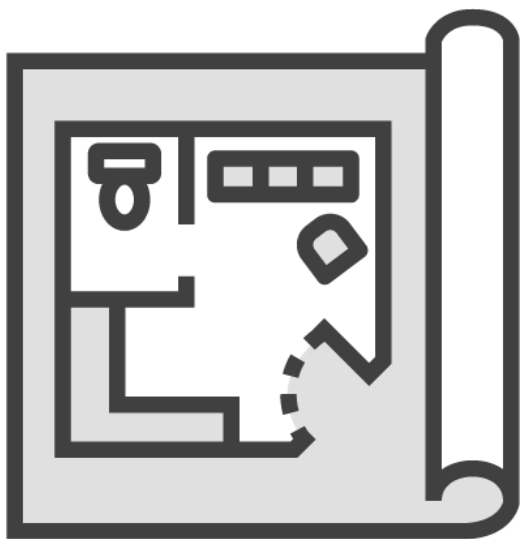
el servidor puede extender la funcionalidad del cliente

### Interfaz uniforme

Todos los recursos del servidor tienen un nombre en forma de URL o hipervínculo



## Interfaz Uniforme - Subrestricciones

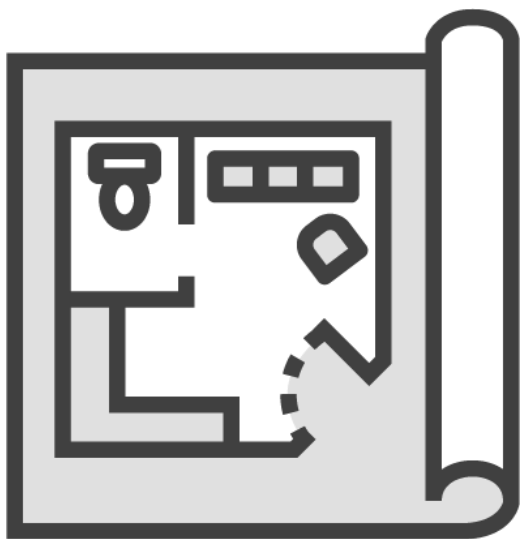


### Identificación de recursos.

- Un recurso esta separado conceptualmente de la representación.
- Media Types de representación: application/json, application/xml, custom, ...



## Interfaz Uniforme - Subrestricciones



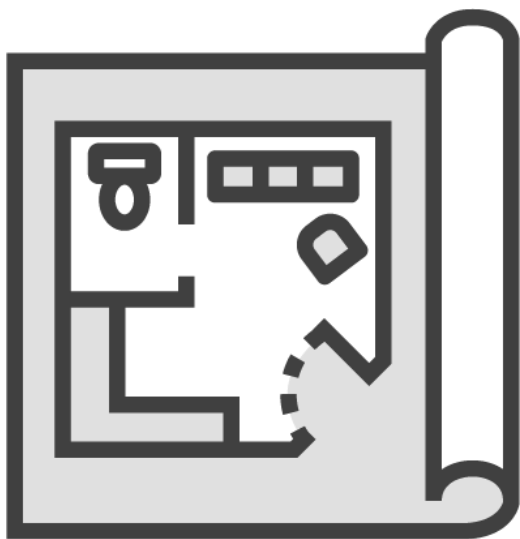
**Manipulación de los recursos atreves de las representaciones**

- Representacion + metadata, deben ser suficientes para modificar o eliminar el recurso





## Interfaz Uniforme - Subrestricciones

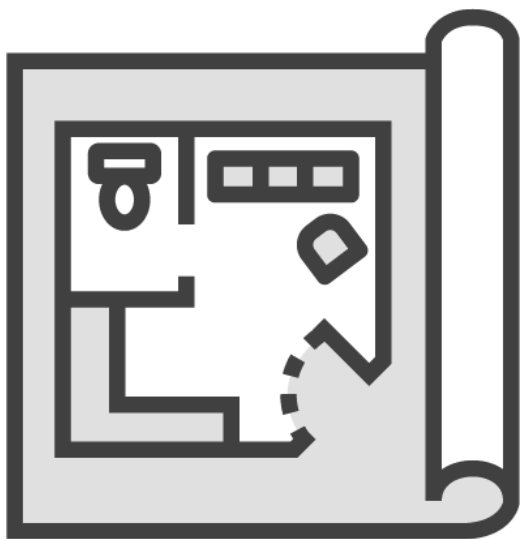


### Mensaje autodescriptivo

- Cada mensaje debe incluir información suficiente para describir cómo procesar el mensaje



## Interfaz Uniforme - Subrestricciones



### Hypermedia as the Engine of Application State (HATEOAS)

- Hypermedia es una generalización de Hypertext (links)
- Indica como consumir y utilizar la API
- Permite la auto-documentación de la API



**Un sistema solo se considera RESTful cuando se adhiere a todas las restricciones requeridas**

La mayoría de las API "RESTful" no son realmente RESTful ...

... pero eso no los hace malos APIs, siempre y cuando se entienda las posibles compensaciones



## Modelo de madurez de Richardson

### Nivel 0 (Swamp of POX)

El protocolo HTTP se utiliza para la interacción remota.

... el resto del protocolo no se usa como debería ser.

Implementaciones de estilo RPC (SOAP, a menudo vistas cuando se usa WCF)

### Nivel 1 (Recursos)

Cada recurso se asigna a un URI

Los métodos HTTP no se usan como deberían ser

Resultados en complejidad reducida



## Modelo de madurez de Richardson

### Nivel 2 (Verbos)

Se utilizan correctamente los verbos HTTP.

Se utilizan correctamente los códigos de estado.

Remueve las variaciones innecesarias.

### Nivel 3 (Hypermedia)

La API tiene soporte de Hypermedia as the Engine of Application State (HATEOAS)

Autodocumentacion



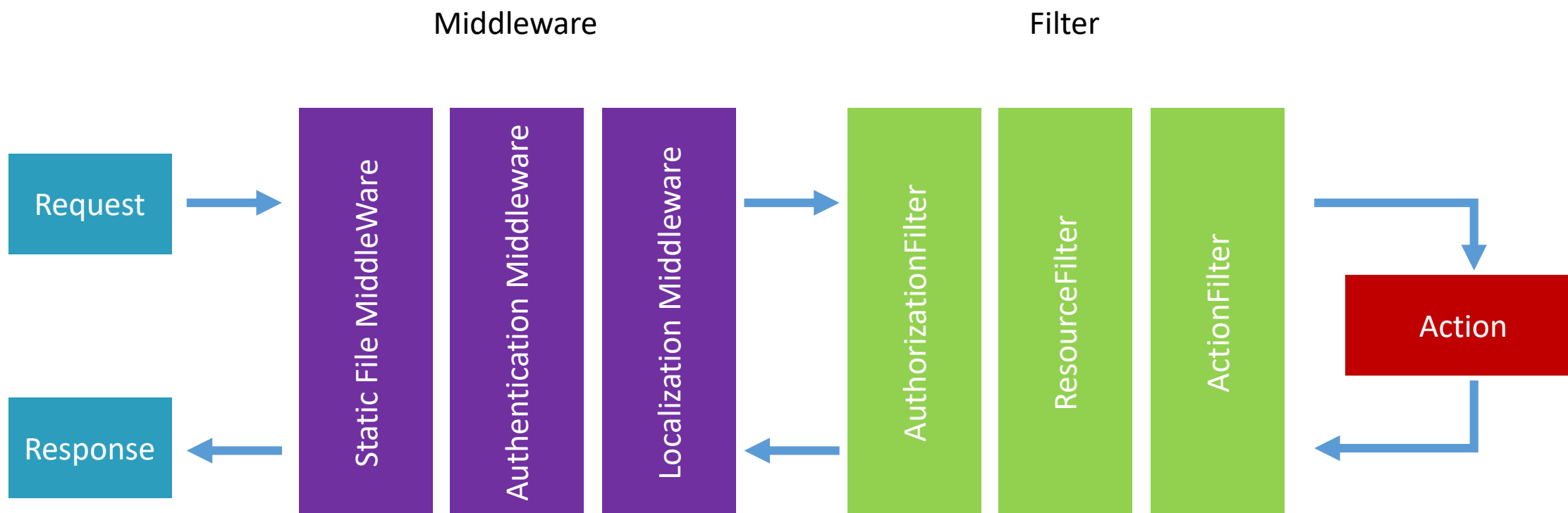
## Middleware

Middleware es un software que se ensambla en un canal (pipe) de aplicaciones para manejar solicitudes y respuestas





## Middleware





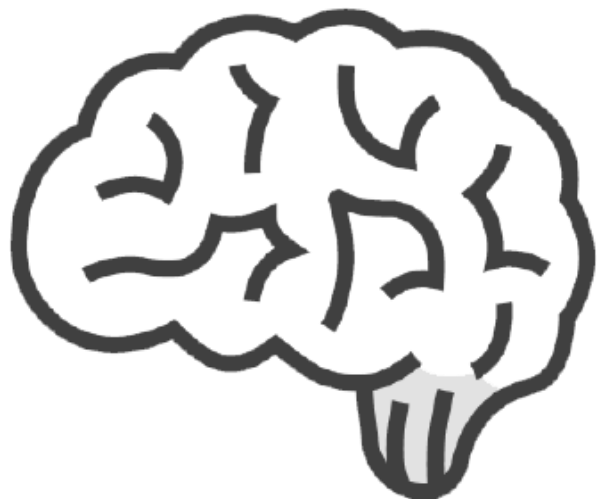
## ASP.NET Core para crear API RESTful



Es un framework open-source, cross-platform que nos permite la construcción de aplicaciones modernas conectadas a internet



## ASP.NET Core para crear API RESTful

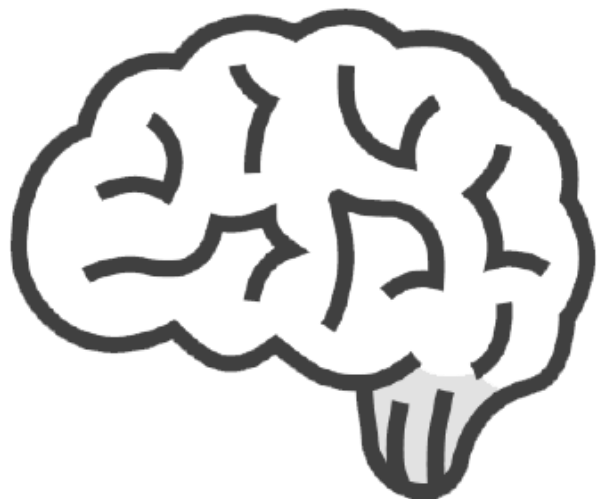


El middleware ASP.NET Core MVC proporciona un marco para crear APIs y aplicaciones web usando el patrón Model-View-Controller

... ¡pero no obtenemos una API RESTful de la caja!



## Patron Model-View-Controller



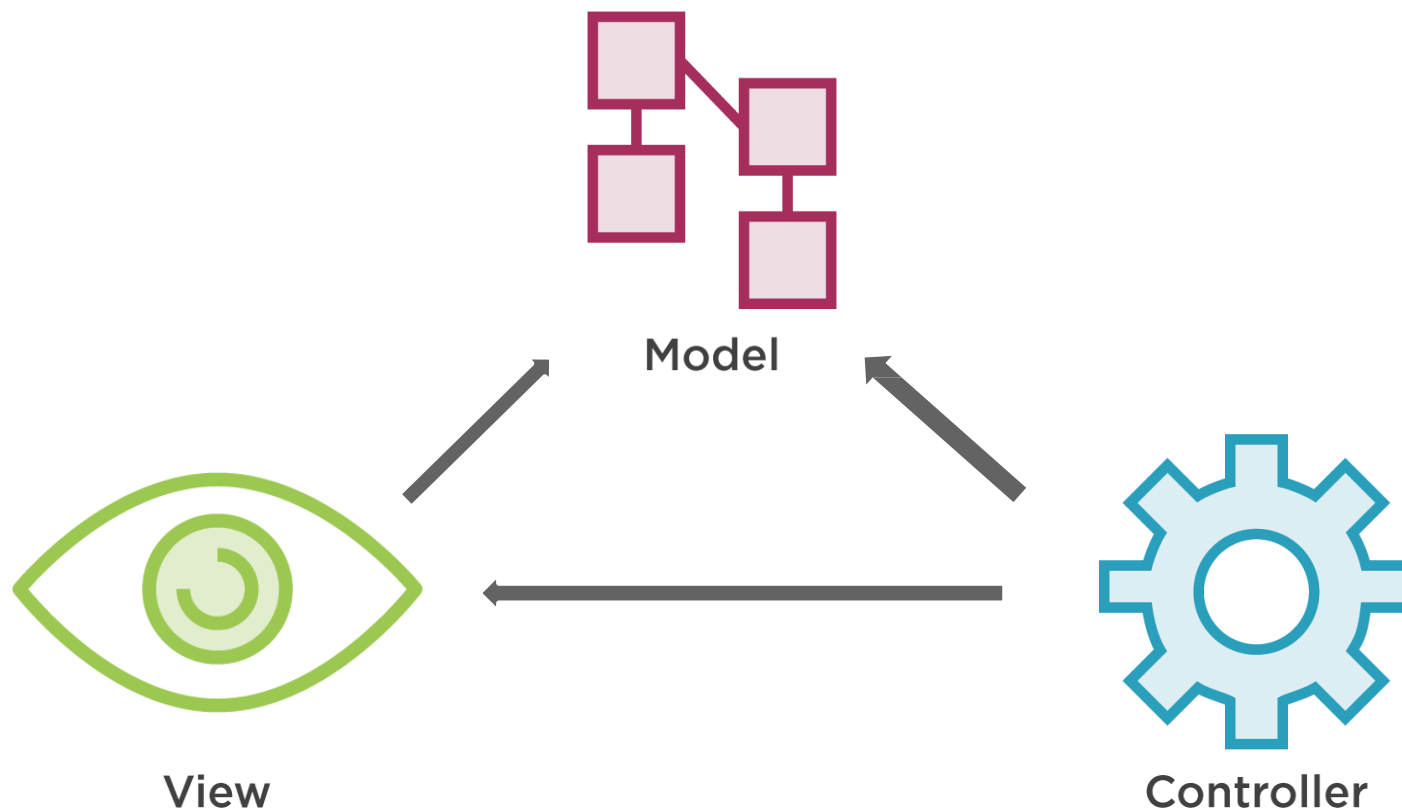
**Patrón arquitectónico para la implementación de interfaces de usuario.**

**Fomenta el bajo acoplamiento y la separación de la responsabilidades.**

**¡No es la arquitectura completa de la aplicación!**

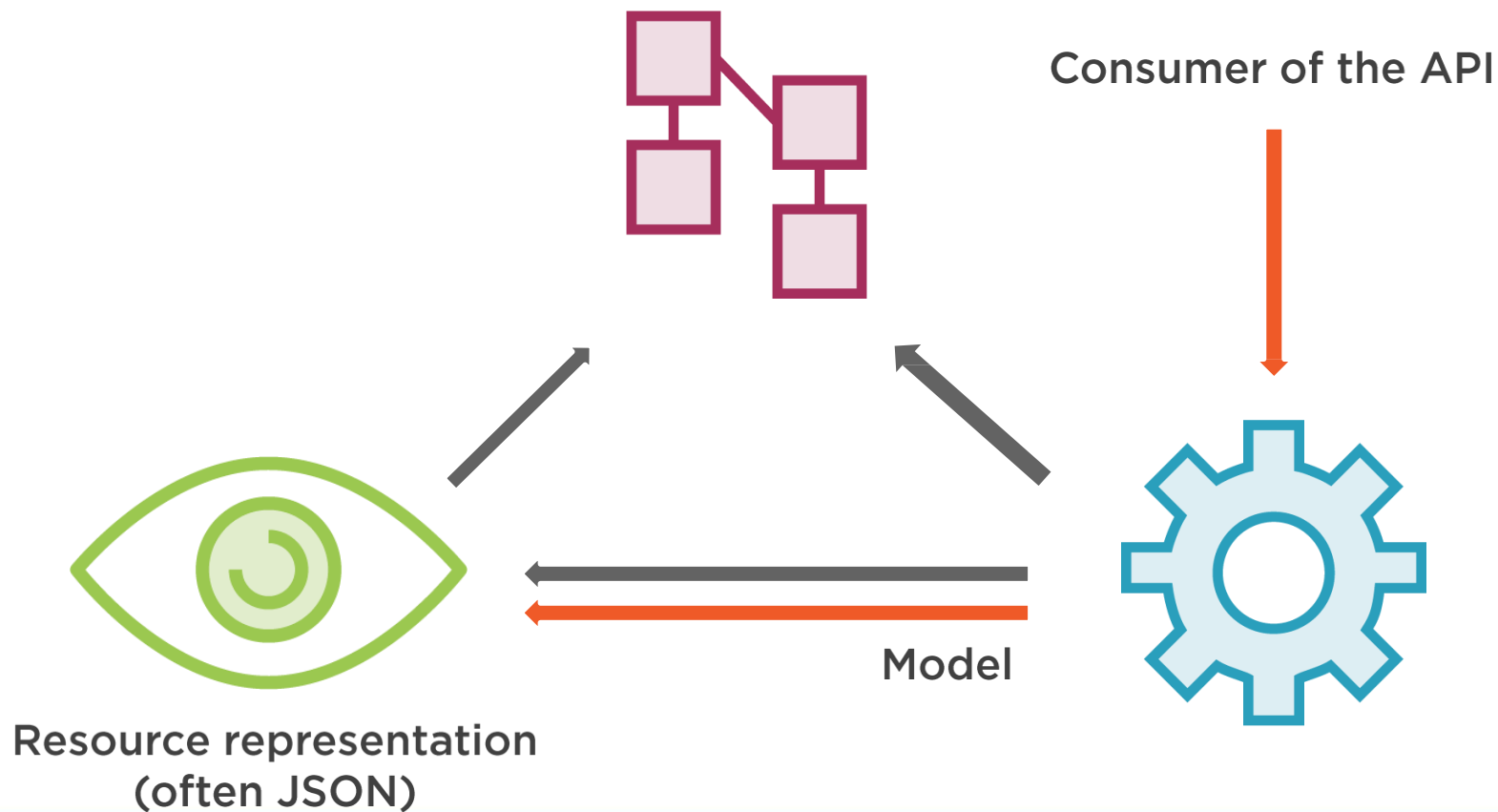


## Patron Model-View-Controller (API)





## Patron Model-View-Controller (API)







## Lineamientos para el nombramiento de recursos



**Sustantivos: cosas, no acciones.**

- ~~api/getauthors~~
- **GET** api/authors
- **GET** api/authors/{authorId}

Transmitir significado al elegir sustantivos



## Lineamientos para el nombramiento de recursos



Seguir este principio para la predictibilidad

- ~~api/something/somethingelse/employees~~
- api/employees
- ~~api/id/employees~~
- api/employees/{employeeId}



## Lineamientos para el nombramiento de recursos



### Representar la jerarquía al nombrar recursos

- [api/authors/{authorId}/books](#)
- [api/authors/{authorId}/books/{bookId}](#)



## Lineamientos para el nombramiento de recursos



Filters, sorting orders, ... no son recurso

- ~~api/authors/orderby/name~~
- api/authors?orderby=name



## Lineamientos para el nombramiento de recursos



A veces, las llamadas al estilo RPC no se asignan fácilmente a nombres de recursos pluralizados

- ~~api/authors/{authorId}/pagetotals~~
- ~~api/authorpagetotals/{id}~~
- api/authors/{authorId}/totalamountofpages



## Lineamientos para el nombramiento de recursos



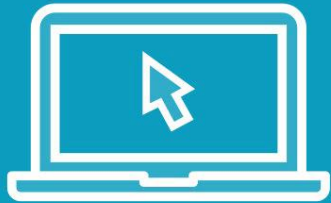
### Representar la jerarquía al nombrar recursos

- [api/authors/{authorId}/books](#)
- [api/authors/{authorId}/books/{bookId}](#)





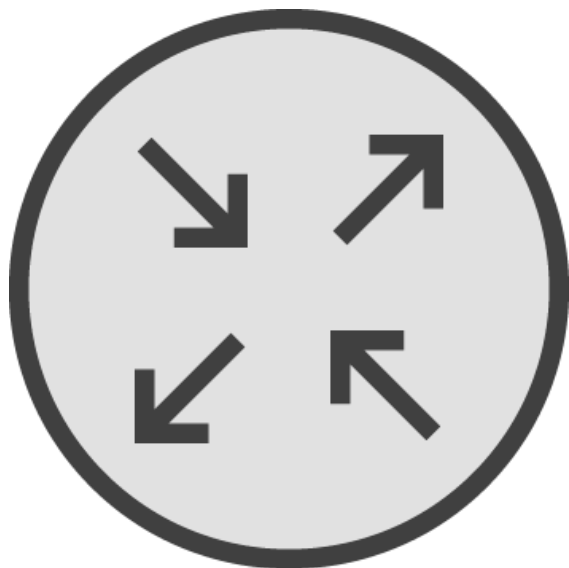
Demo



Creación del Web API.



## Trabajando con Routing



El enrutamiento coincide con un URI de solicitud a una acción en un controlador.



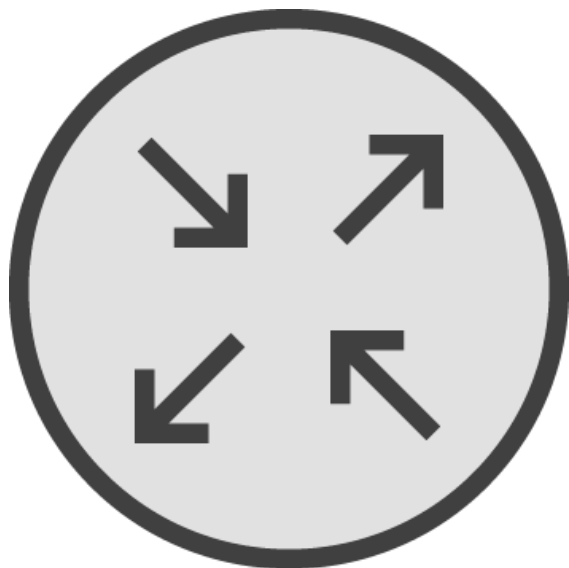
## Trabajando con Routing

```
app.UseMvc(routes =>{  
    routes.MapRoute(  
        name: "default",  
        template:  
            "{controller=Values}/{action=Index}/{id?}");  
    });
```

El equipo de ASP.NET Core recomienda utilizar enrutamiento basado en atributos para las API



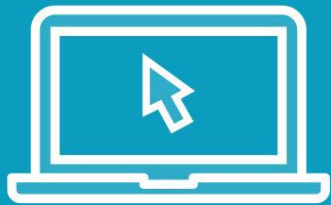
## Trabajando con Routing



Utilizar los atributos a nivel de controlador y de acción: [Ruta], [HttpGet], ...



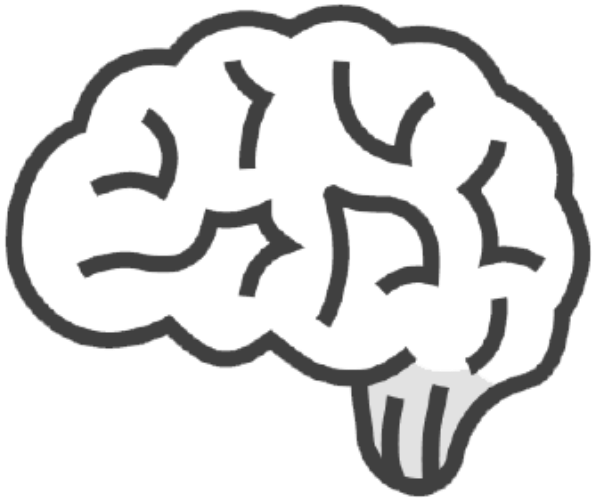
Demo



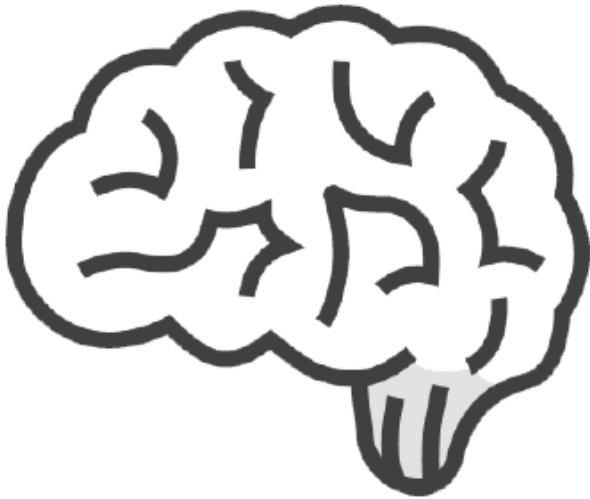
Routing



Exponiendo métodos CRUD de la capa de servicios.



**Outer Facing Model**  
**!=**  
**Entity Model**



**Outer Facing Model**

**!=**

**Business Model**

**!=**

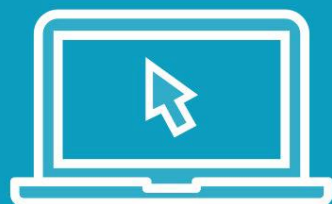
**Entity Model**



# Exponiendo métodos CRUD de la capa de servicios.



Demo

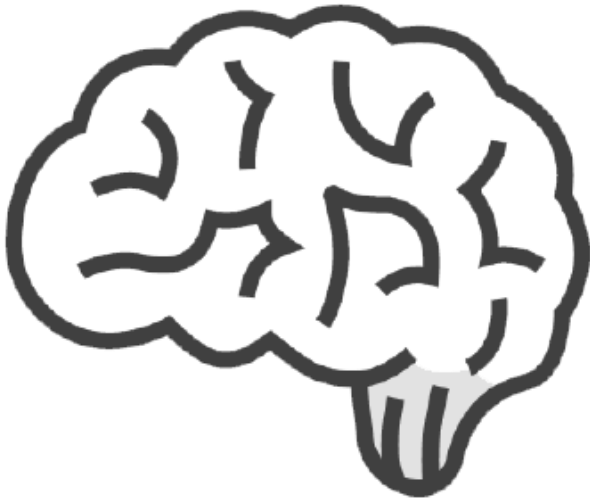


Obteniendo una colección de recursos





## La importancia de los códigos de estado



**Los códigos de estado le dicen al consumidor de la API**

- Si la solicitud funcionó o no como se esperaba
- El motivo de la falla



## La importancia de los códigos de estado

### Level 200 - Success

200 – Ok

201 – Created

204 – No content

### Level 400 – Client Mistakes

400 – Bad request

401 – Unauthorized

403 – Forbidden

404 – Not found



## La importancia de los códigos de estado

### Level 400 –Client Mistakes

405 – Method not allowed

406 – Not acceptable

409 - Conflict

415 – Unsupported media type

422 – Unprocessable entity

### Level 500 Server Mistakes

500 – Internal server error



# Exponiendo métodos CRUD de la capa de servicios.



Demo



Códigos de estado



## Errores Versus Faults

### Errores

El consumidor pasa datos inválidos a la API, y la API rechaza

Los códigos de estado del nivel 400 no contribuyen a la disponibilidad de la API

### Faults

La API no puede devolver una respuesta a una solicitud válida

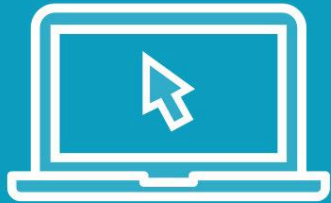
Códigos de estado del nivel 500  
Contribuye a la disponibilidad de API



# Exponiendo métodos CRUD de la capa de servicios.



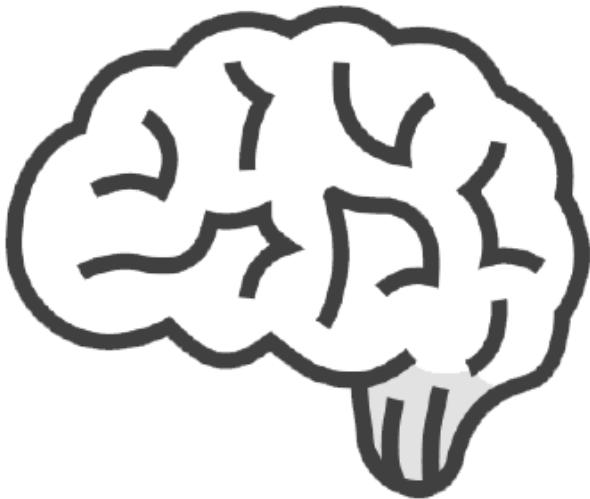
Demo



Gestión de errores



## Formateadores y Negociación de Contenido



Seleccionar la mejor representación para una respuesta dada cuando hay múltiples representaciones disponibles

El tipo de medio se pasa a través “Accept header” de la solicitud

- application/json
- application/xml
- ...



## Formateadores y Negociación de Contenido



Output formatter

Deals with output  
Media type: accept header



Input formatter

Deals with input  
Media type: content-type header





# Exponiendo métodos CRUD de la capa de servicios.



Demo



Formato y contenido



## Propósito de un token de seguridad

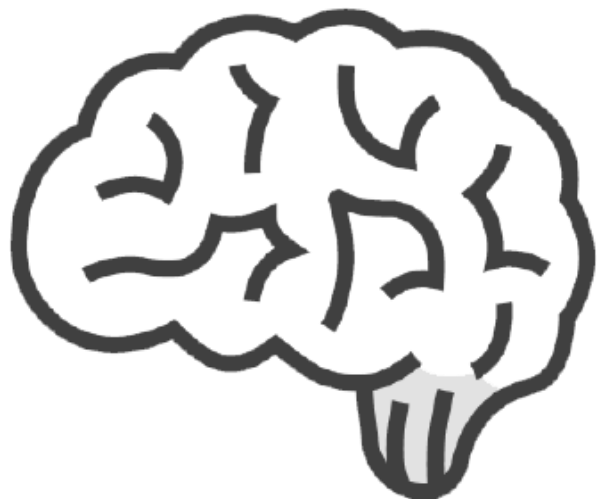


**Los tokens de seguridad son estructuras de datos (protegidas)**

- Contiene información sobre el emisor y el solicitante (claims).
- Firmado (a prueba de manipulaciones y autenticidad)
- Normalmente contienen un tiempo de caducidad



## Propósito de un token de seguridad



**Un cliente solicita un token**

**Un emisor emite un token**

**Un recurso consume un token.**

- Tiene una relación de confianza con el emisor.



## Historia

### **SAML 1.1 / 2.0**

- Basado en XML
- Muchas opciones de cifrado y firma
- Muy expresivo

### **Web simple Token (SWT)**

- Formulario / URL codificado
- Sólo firmas simétricas

### **JSON Web Token (JWT)**

- Codificado en JSON
- firmas simétricas y asimétricas (HMACSHA256-384, ECDSA, RSA)
- cifrado simétrico y asimétrico (RSA, AES / CGM)
- (El nuevo estándar)



# Estructura de JWT (header, payload y signature).



## JSON Web Token

**En camino a la estandarización oficial.**

- <http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html>

### **Header**

- metadata
- algoritmos y claves utilizadas

### **Claims (Payload)**

- Issuer (iss)
- Audience (aud)
- IssuedAt (iat)
- Expiration (exp)
- Subject (sub)
- ...claims definidos por la aplicación



# Estructura de JWT (header, payload y signature).



## Estructura Structure

Header

```
{  
  "typ": "JWT",  
  "alg": "HS256"  
}
```

Claims

```
{  
  "iss": "http://myIssuer",  
  "exp": "1340819380",  
  "aud": "http://myResource",  
  "sub": "alice",  
  
  "client": "xyz",  
  "scope": ["read", "search"]  
}
```

eyJhbGciOiJIub251In0.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMD.4MTkzODAsDQogImh0dHA6Ly9leGFt

Header

Claims

Signature



# Exponiendo métodos CRUD de la capa de servicios.



Demo



Implementación nativa de JWT en Net Core.

Configuración de servicios con JWT

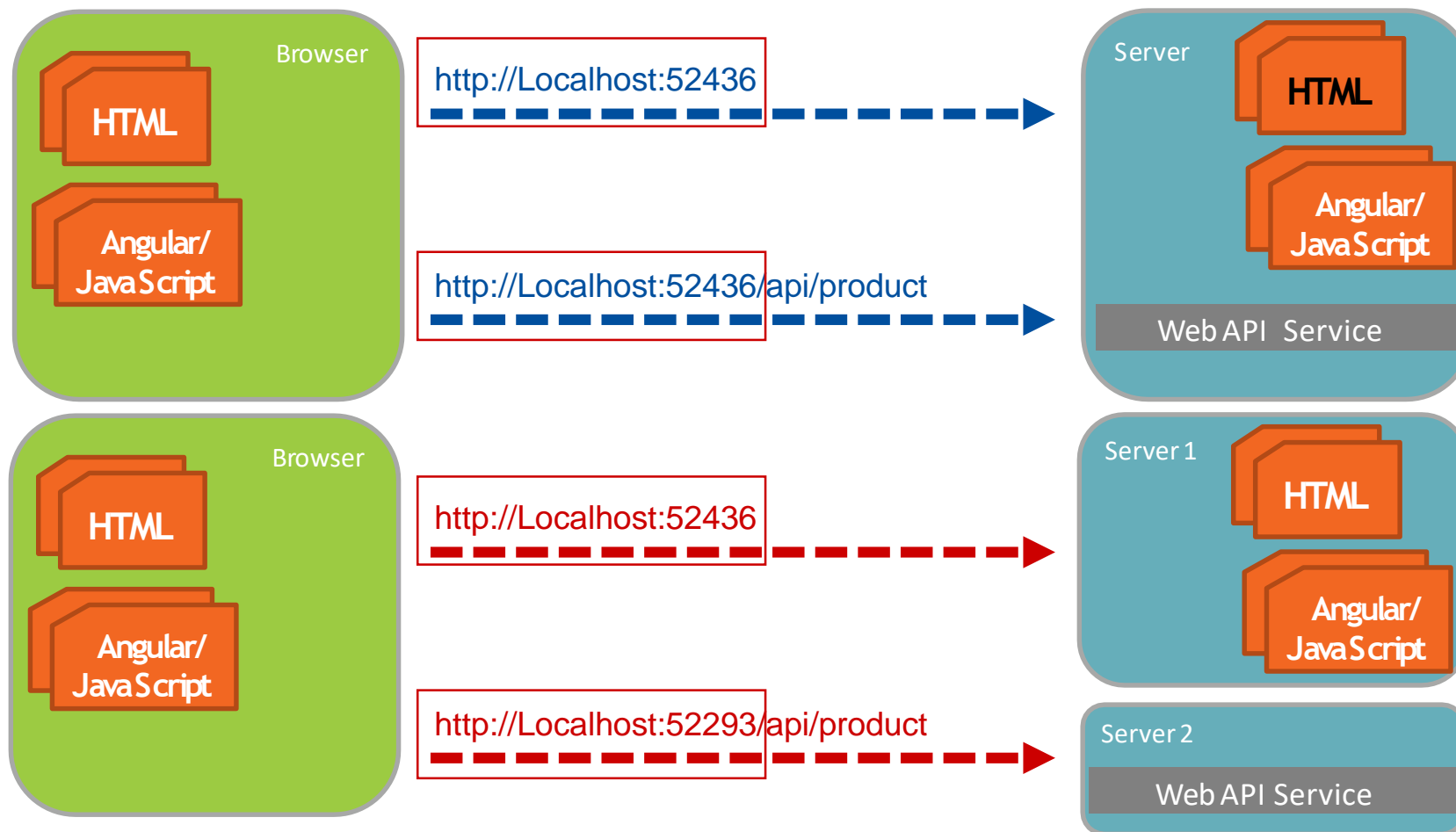


**CORS ... permite que muchos recursos ... en una página web se soliciten desde otro dominio fuera del dominio desde el cual se originó el recurso.**





# Configuración de Cross-Origin Requests (CORS).

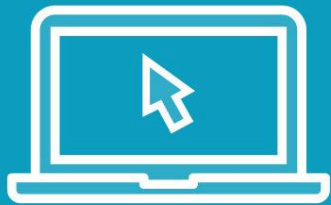




# Configuración de Cross-Origin Requests (CORS).



Demo



CORS



GALAXY  
TRAINING