.NET CORE

3. HTTP API REST

plain concepts

51

¿Qué es un API REST?

API REST: Definición

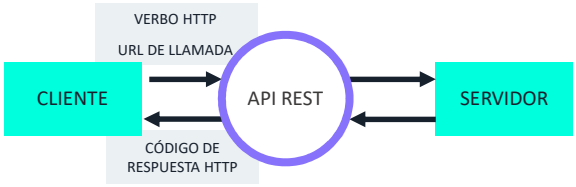
- *API (Application Programming Interface)* será el contrato que nuestro servidor le ofrezca a todos sus clientes.
- *REST (Representational State Transfer)* es un estilo de arquitectura para diseñar aplicaciones en la red.
  - Cada **petición HTTP contiene toda la información necesaria para ejecutarla**, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla.
  - Se compone de una lista de reglas que se deben cumplir en el diseño de la arquitectura de una API.
  - Hablaremos de **servicios web restful** si cumplen la arquitectura REST

52

¿Qué es un API REST?

API REST: ¿Cómo funciona?

- Las llamadas al API se implementan como peticiones HTTP
  - La URL representa el recurso
    - https://reqres.in/api/users?page=2
  - El verbo HTTP la acción
    - GET, POST, PUT, DELETE
  - El código de estado HTTP representa el resultado
    - 200 (OK), 201 (CREATED), 404 (NOT FOUND), 500 (INTERNAL SERVER ERROR)
- Reglas de la Arquitectura REST
  - Interfaz uniforme
  - Peticiones sin estado
  - Cacheable
  - Separación de cliente y servidor



53

¿Qué es un API REST?

Interfaz uniforme

- La interfaz de basa en recursos (por ejemplo el recurso Empleado (Id, Nombre, Apellido, Puesto, Sueldo))
- El servidor manda los datos como se los pidan (json, xml...) pero lo que tenga en su interior (BBDD por ejemplo) para el cliente es transparente
- La información del recurso que le llega el cliente podrá ser utilizado en otras operaciones:
  - Suponiendo que tenga permisos
- Procurar una API sencilla y jerárquica y con ciertas reglas: uso de nombres en plural

Peticiones sin estado

- http es un protocolo sin estado
  - GET {baseurl}/empleados/1234
  - DELETE {baseurl}/empleados/1234
- En la segunda petición hemos tenido que incidar el identificador del recurso que queremos borrar
- El servidor no guardaba los datos de la consulta previa que tenía el cliente en partícular.
- Una petición del tipo DELETE mi_url/empleado debe dar error, ¡falta el id y el servidor no lo conoce!

54

¿Qué es un API REST?

Cacheable

- En la web los clientes pueden cachear las respuestas del servidor
- Las respuestas se deben marcar de forma implícita o explícita como cacheables o no.
- En futuras peticiones, el cliente sabrá si puede reutilizar o no los datos que ya ha obtenido.
- Si ahorramos peticiones, mejoraremos la escabilidad de la aplicación y el rendimiento en cliente (evitamos principalmente la latencia).

Separación de cliente y servidor

- El cliente y servidor están separados, su unión es mediante la API.
- Los desarrollos en frontend y backend se hacen por separado, teniendo en cuenta la API.
- Mientras la interfaz no cambie, podremos cambiar el cliente o el servidor sin problemas.

55

¿Qué es un API REST?

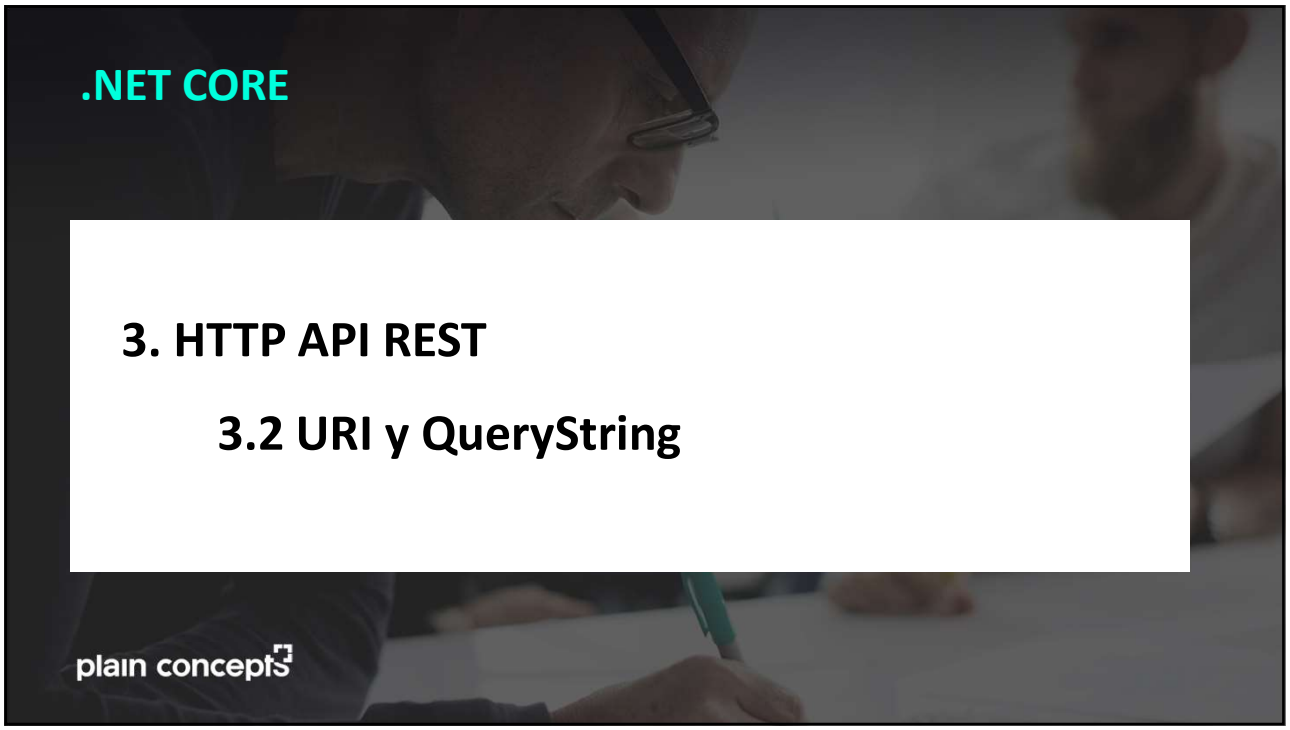
Cacheable

- En la web los clientes pueden cachear las respuestas del servidor
- Las respuestas se deben marcar de forma implícita o explícita como cacheables o no.
- En futuras peticiones, el cliente sabrá si puede reutilizar o no los datos que ya ha obtenido.
- Si ahorramos peticiones, mejoraremos la escabilidad de la aplicación y el rendimiento en cliente (evitamos principalmente la latencia).

Separación de cliente y servidor

- El cliente y servidor están separados, su unión es mediante la API.
- Los desarrollos en frontend y backend se hacen por separado, teniendo en cuenta la API.
- Mientras la interfaz no cambie, podremos cambiar el cliente o el servidor sin problemas.

56



57

URIs, URLs y QueryString

URI

LA URI es el recurso, por lo tanto, es la información a la que queremos acceder o que queremos modificar o borrar, **independientemente de su formato**.

Las URL (Uniform Resource Locator) son un tipo de URI (Uniform Resource Identifier) que además de permitir **identificar de forma única el recurso**, nos permite localizarlo para poder acceder a él o compartir su ubicación.

URL: {protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}

Reglas de una URI:

- Los nombres de URI no deben implicar una acción, por lo tanto, debe evitarse usar verbos en ellos.
- Deben ser únicas, no debemos tener más de una URI para identificar un mismo recurso.
- Deben ser independiente de formato.
- Deben mantener una jerarquía lógica.
- Los filtrados de información de un recurso no se hacen en la URI.

58

URIs, URLs y QueryString

Las URIs no deben implicar acciones y deben ser únicas

Por ejemplo, la URI **/facturas/234/editar** sería incorrecta ya que tenemos el verbo editar en la misma.

Para el recurso factura con el identificador 234, la siguiente URI sería la correcta, independientemente de que vayamos a editarla, borrarla, consultarla o leer sólo uno de de sus conceptos: **/facturas/234**

Las URIs deben ser independientes de formato

Por ejemplo, la URI **/facturas/234.pdf** no sería una URI correcta, ya que estamos indicando la extensión pdf en la misma.

Para el recurso factura con el identificador 234, la siguiente URI sería la correcta, independientemente de que vayamos a consultarla en formato pdf, epub, txt, xml o json: **/facturas/234**

59

URIs, URLs y QueryString

Las URIs deben mantener una jerarquía lógica

Por ejemplo, la URI **/facturas/234/cliente/007** no sería una URI correcta, ya que no sigue una jerarquía lógica.

Para el recurso factura con el identificador 234 del cliente 007, la siguiente URI sería la correcta:
/clientes/007/facturas/234

Filtrados y otras operaciones.

Para filtrar, ordenar, paginar o buscar información en un recurso, debemos hacer una consulta sobre la URI, utilizando parámetros HTTP en lugar de incluirlos en la misma.

Por ejemplo, la URI **/facturas/orden/desc/fecha-desde/2007/pagina/2** sería incorrecta ya que el recurso de listado de facturas sería el mismo pero utilizaríamos una URI distinta para filtrarlo, ordenarlo o paginarlo.

La URI correcta en este caso sería:

/facturas?fecha-desde=2007&orden=DESC&pagina=2

Lo que hay a partir de la ? se denomina **QueryString**

60

URIs, URLs y QueryString

Las URIs deben mantener una jerarquía lógica

Por ejemplo, la URI **/facturas/234/cliente/007** no sería una URI correcta, ya que no sigue una jerarquía lógica.

Para el recurso factura con el identificador 234 del cliente 007, la siguiente URI sería la correcta:
/clientes/007/facturas/234

Filtrados y otras operaciones.

Para filtrar, ordenar, paginar o buscar información en un recurso, debemos hacer una consulta sobre la URI, utilizando parámetros HTTP en lugar de incluirlos en la misma.

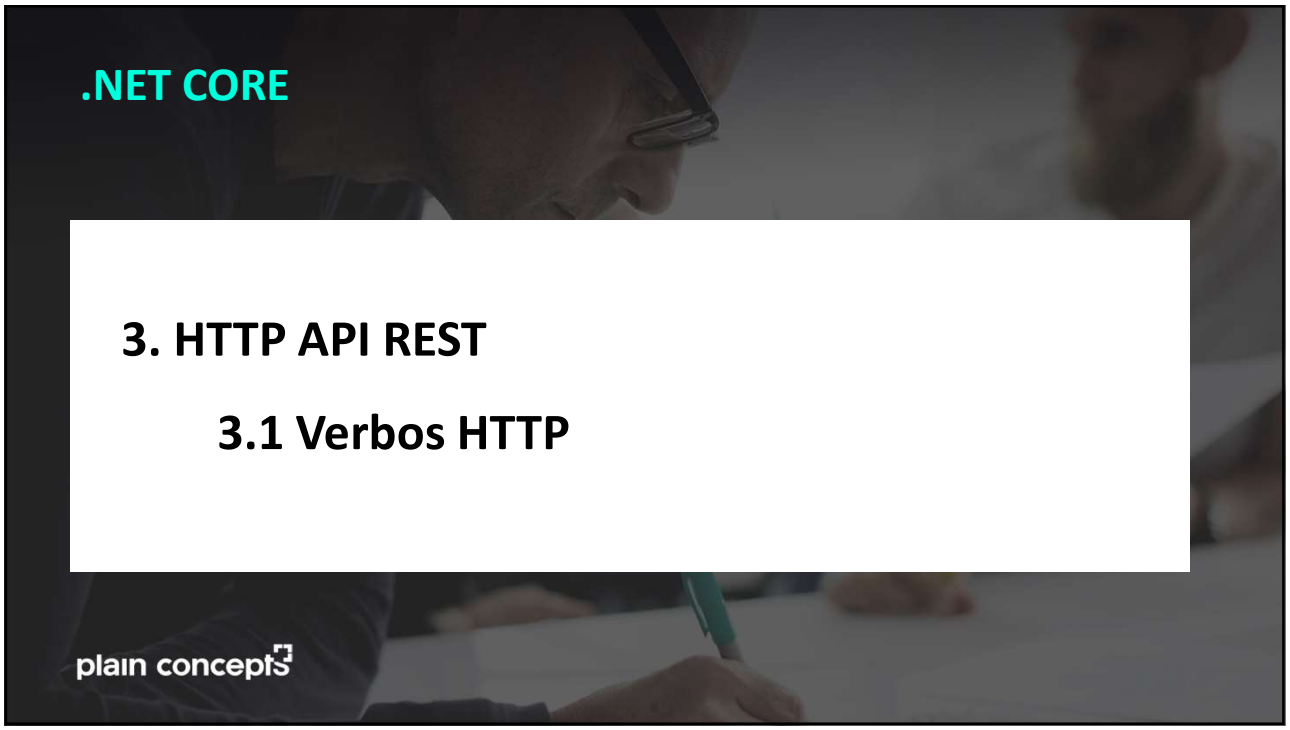
Por ejemplo, la URI **/facturas/orden/desc/fecha-desde/2007/pagina/2** sería incorrecta ya que el recurso de listado de facturas sería el mismo pero utilizaríamos una URI distinta para filtrarlo, ordenarlo o paginarlo.

La URI correcta en este caso sería:

/facturas?fecha-desde=2007&orden=DESC&pagina=2

Lo que hay a partir de la ? se denomina **QueryString**

61



62

Verbos HTTP

Verbos HTTP

Existen cuatros Verbos HTTP que indican que operaciones vamos a llevar a cabo:

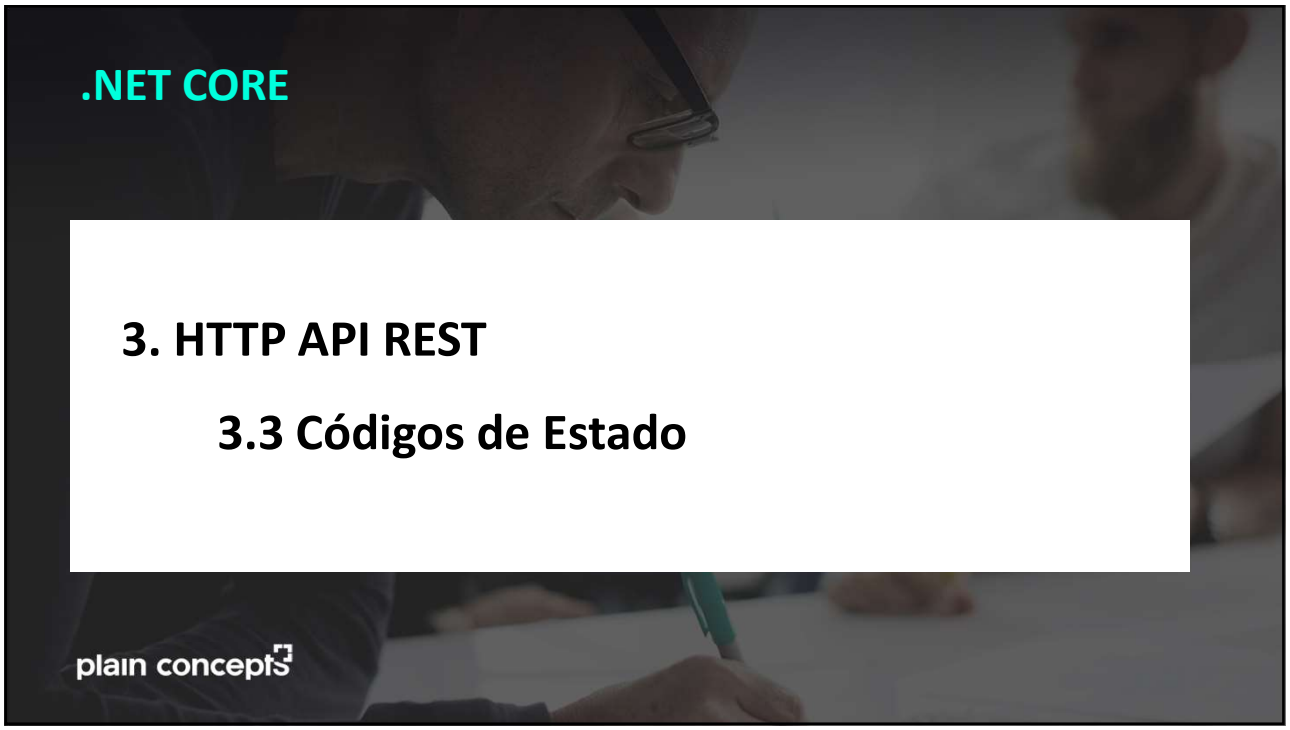
- GET: Para consultar y leer recursos
- POST: Para crear recursos
- PUT: Para editar recursos
- DELETE: Para eliminar recursos.
- PATCH: Para editar partes concretas de un recurso. (poco usado)

Ejemplos:

- GET /facturas Nos permite acceder al listado de facturas
- POST /facturas Nos permite crear una factura nueva
- GET /facturas/123 Nos permite acceder al detalle de una factura
- PUT /facturas/123 Nos permite editar la factura, sustituyendo la totalidad de la información anterior por la nueva.
- DELETE /facturas/123 Nos permite eliminar la factura

Es un error emplear solo métodos GET y POST para todas las operaciones de nuestra API.

63



64

Códigos de Estado

Códigos de Estado HTTP

Si tenemos una petición PUT /facturas/123 y la respuesta que nos devuelven es:
Status Code 200
Content:

```
{
        success: false,
        code:   754,
        errormessage:   "datos insuficientes"
}
```

Esto está devolviendo un código 200 (Respuesta correcta), y en el cuerpo de la respuesta el error. Inconvenientes:

- Esto no es una respuesta standard ni es Restfull
- El cliente se tendría que adaptar a este tipo de respuestas.
- Tenemos que mantener nuestros propios códigos en el servidor

No hay que reinventar la rueda con las respuestas en un **API** que va por **HTTP**, **hay que respetar el protocolo**. Para ello HTTP dispone de unos códigos de respuesta para devolver lo que ha ocurrido en la llamada:

- https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP

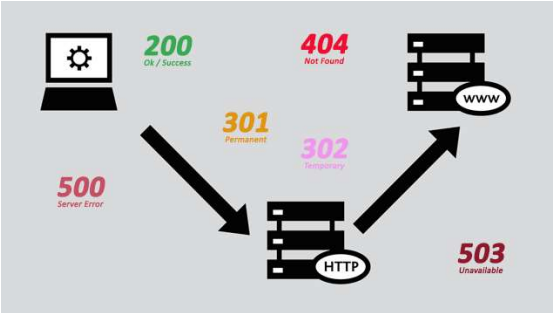
65

Códigos de Estado

Códigos de Estado HTTP

HTTP tiene un abánico muy amplio de códigos de error y su codificación genérica es:

- 100-199: Respuestas informativas
- 200-299: Peticiones correctas
- 300-399: Redirecciones
- 400-499: Errores del cliente
- 500-599: Errores del servidor



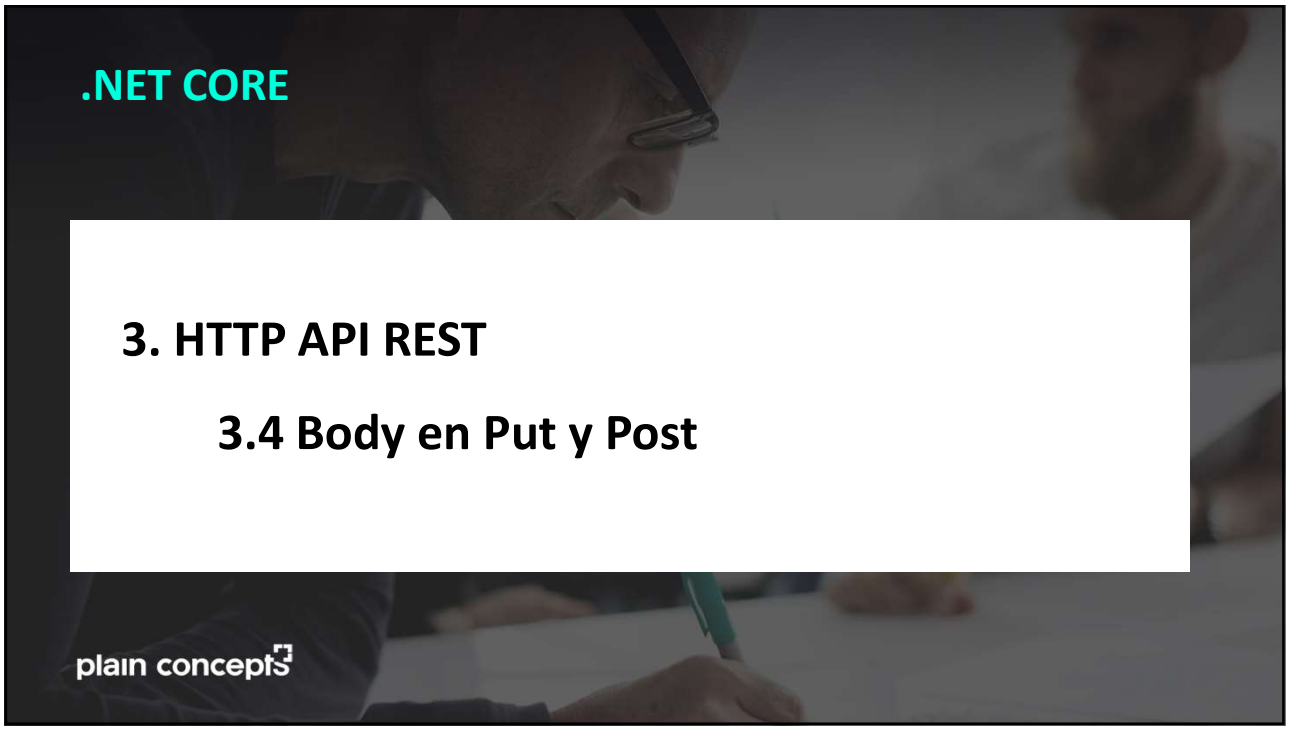
66

Códigos de Estado

Códigos de Estado HTTP

CÓDIGO	SIGNIFICADO	EXPLICACIÓN
200	OK	Todo está funcionado
201	OK	Nuevo recurso ha sido creado
204	OK	El recurso ha sido borrado satisfactoriamente
400	Bad Request	La petición es inválida o no puede ser servida.
401	Unauthorized	La petición requiere de la autenticación del usuario
403	Forbidden	El servidor entiende la petición pero la rechaza o el acceso no está permitido
404	Not found	No encontrado.
500	Internal Server Error	Error en el Servidor

67



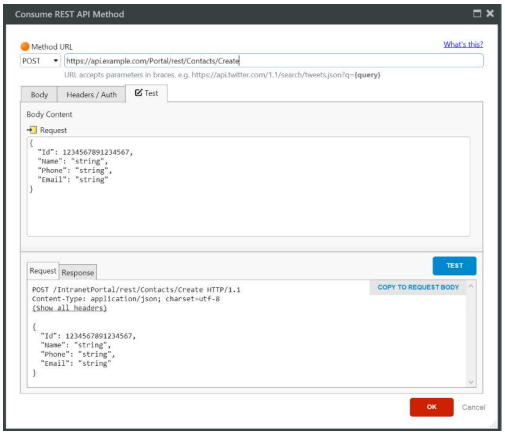
68

Body en Put y Post

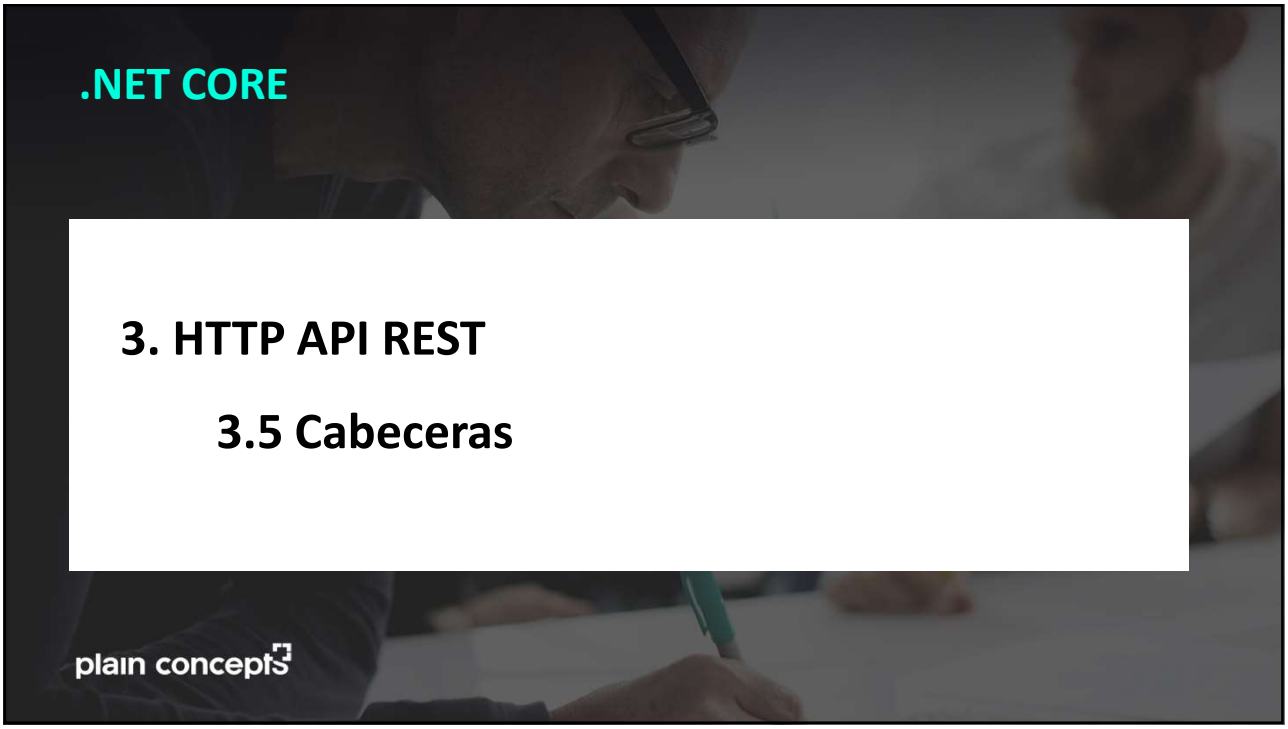
Body

Las peticiones y respuestas HTTP tienen una estructura similar donde comparten un campo llamado cuerpo que es opcional. Este campo:

- Es el que lleva la información del recurso que se va a modificar.
- No puede usarse en peticiones GET. Solo se usa en POST y PUT. Muy raramente en DELETEs.



69



70

Cabeceras HTTP

Cabeceras HTTP

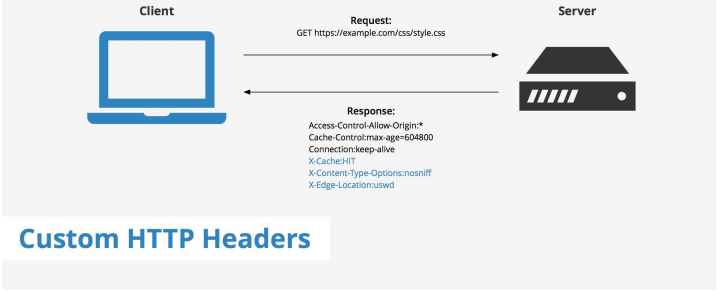
Las peticiones y respuestas HTTP también tienen otra peculiaridad: las cabeceras.

- Las cabeceras permiten al cliente y servidor enviar información adicional junto a una petición.
- Una cabecera de petición esta compueta por su nombre (no sensible a las mayusculas) seguido de dos puntos ':', y a continuación su valor (sin saltos de línea)
- Existen muchas cabeceras standard, pero se pueden añadir cabeceras propietarias usando el sufijo 'x-'.

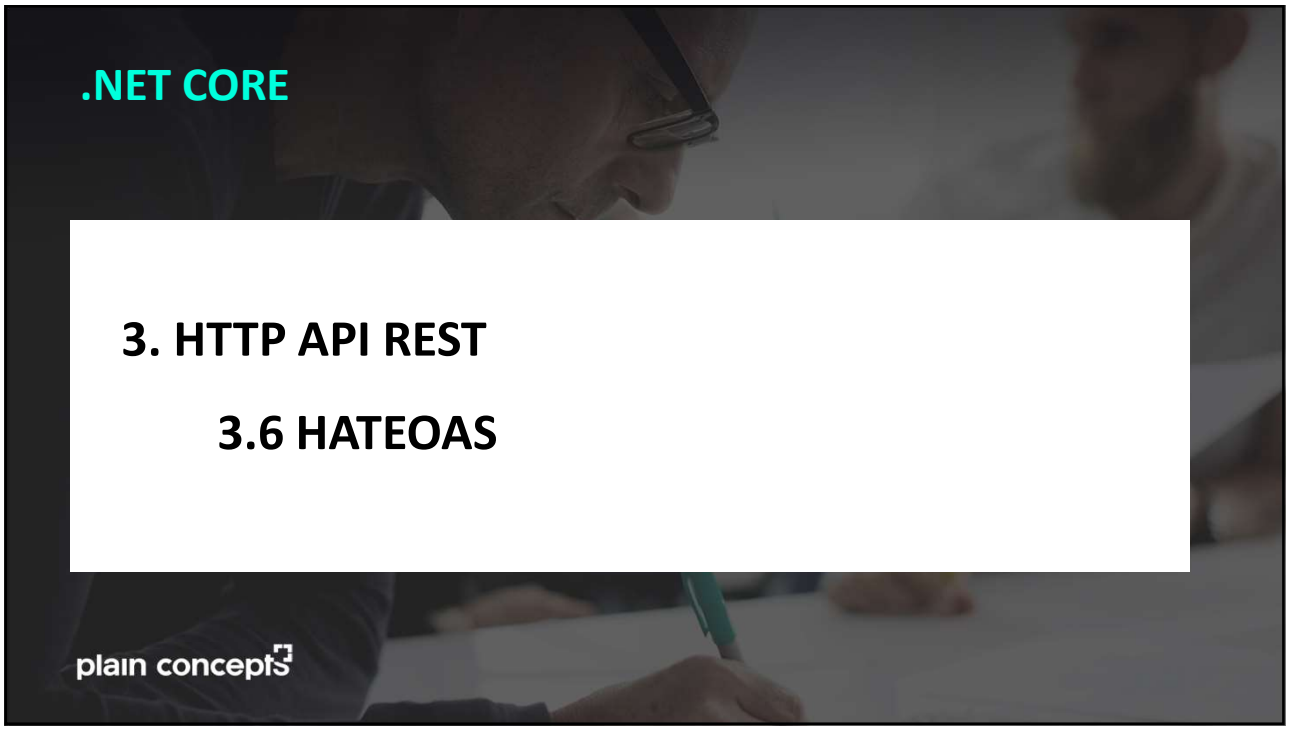
Existen cabeceras de todo tipo. Las Podemos ver en: https://developer.mozilla.org/es/docs/Web/HTTP/Headers

Quizá la cabecera más conocida sea la de Content-Type, que nos indica como queremos recibir la información del API.

- Content-type: application/json



71



72

HATEOAS

HATEOAS (Hypermedia as the Engine of Application State)

Otra restrición de las APIs REST es que deben ser HATEOAS. O lo que es lo mismo, deben disponer de enlaces a otros recursos que referenciamos.
Si por ejemplo pedimos una factura y esa factura tiene un cliente, en nuestra petición debería venir la llamada al recurso de ese cliente:

```
{
    id: 344,
    amount: 525,
    orderId: 266,
    client: {
        id: 155,
        href: '/clients/155'
    }
}
```

73

Demo

POSTMAN

plain concepts

74