UA . MASTER MOVILES

MÁSTER UNIVERSITARIO EN DESARROLLO DE SOFTWARE PARA DISPOSITIVOS MÓVILES

PROGRAMACIÓN HIPERMEDIA PARA DISPOSITIVOS MÓVILES

Servicios REST

CONTENIDOS

- Fundamentos REST
- Tipos de peticiones HTTP
- Invocación de servicios
- Aclaraciones

ACCESO A SERVICIOS WEB DESDE EL MÓVIL

- Normalmente accedemos a la web desde un **navegador**:
  → Obtiene documentos (HTML) para mostrar al usuario.
- Acceso desde una **aplicación móvil**:
  → Necesitamos obtener información, no presentación.
  → Resulta adecuado acceder a servicios web.
- Definición de servicio:
  - Interfaz que da acceso a un módulo de funcionalidad.
- Servicio web:
  - Servicio al que se accede mediante protocolos web (HTTP, XML)
- Tipos de servicios
  - SOAP: Muy pesado. Adecuado para integración de aplicaciones
  - RESTful: Ligero y sencillo. Adecuado para web y móviles

FUNDAMENTOS REST

- REST (*REpresentational State Transfer*): Son un conjunto de restricciones que, aplicadas al diseño de un sistema, crean un estilo arquitectónico caracterizado por:
  - Debe ser un sistema cliente-servidor.
  - Tiene que ser sin estado.
  - Tiene que soportar caché.
  - Tiene que ser un sistema uniformemente accesible (a través de URIs).
  - Tiene que ser un sistema por capas (escalabilidad)
- Estas restricciones **no dictan qué tipo de tecnología** utilizar.
- Podemos utilizar las infraestructuras de red existentes.
- Un ejemplo de sistema RESTful: la web estática.

RECURSOS

- Un recurso es “cualquier cosa” que pueda ser accedido y transferido a través de la red:
  - Es una correspondencia lógica y temporal con un concepto del domino del problema.
  - Ejemplos: una noticia de un periódico, la temperatura de Alicante, un valor de IVA almacenado en una BD, etc.
- Cada uno de los recursos puede ser accedido directamente, y de forma independiente, pero diferentes peticiones podrían “apuntar” al mismo dato.
- La representación de un recurso depende del tipo solicitado por el cliente (tipo MIME).

REPRESENTACIÓN

- Lo que se intercambia entre los consumidores (clientes) y los productores de servicios son representaciones de los recursos.
- Una representación muestra el estado de un dato almacenado en el momento de la petición.
- El “lenguaje” de intercambio de información (representación) con el servicio queda a elección del desarrollador.
- Ejemplos de representaciones comunes son:

Formato	Tipo MIME
Texto plano	text/plain
HTML	text/html
XML	application/xml
JSON	application/json

REPRESENTACIÓN JSON

- JSON, acrónimo de "*JavaScript Object Notation*", es un formato ligero para el intercambio de datos.
- Es un subconjunto de la notación literal de objetos de JavaScript.
- Tipos de datos y notación:
  - Array: entre corchetes `[]`
  - Objetos: entre llaves `{}`
  - Como separador se utiliza la coma "`,`".
  - String o nombres de variables: entre comillas dobles "`"`".
  - También admite números y booleanos.
- Por ejemplo:

```
[
    {"nombre": "Pedro", "edad": 25, "validado": true},
    {"nombre": "Laura", "edad": 22, "validado": false}
]
```

URI

- Una URI (*Uniform Resource Identifier*) en un servicio web RESTful es un hiper-enlace a un recurso y es la única forma de intercambiar representaciones entre clientes y servidores.
- En un sistema REST la URI **no cambia a lo largo del tiempo**.
- Si por ejemplo en nuestro sistema tenemos información de cursos, podríamos acceder a una lista de cursos disponibles mediante la siguiente URL: `http://<domain>/cursos`
- Esto nos podría devolver un documento como el siguiente:

```
<?xml version="1.0"?>
<j:Cursos xmlns:j="http://www.ua.es"
   xmlns:xlink="http://www.w3.org/1999/xlink">
   <Curso id="1" xlink:href="http://www.ua.es/resources/cursos/1"/>
   <Curso id="2" xlink:href="http://www.ua.es/resources/cursos/2"/>
   <Curso id="4" xlink:href="http://www.ua.es/resources/cursos/4"/>
   <Curso id="6" xlink:href="http://www.ua.es/resources/cursos/6"/>
</j:Cursos>
```

UNIFORMIDAD DE LAS INTERFACES

- Un servicio RESTful limita la ambigüedad en el diseño y la implementación restringiendo las operaciones que podemos realizar con los recursos.
- Las operaciones posibles se conocen como CRUD:

```
Create, Retrieve, Update, Delete
```

- La correspondencia de estas acciones con métodos HTTP son:

Acción sobre los datos	Protocolo HTTP equivalente
CREATE	POST
RETRIEVE	GET
UPDATE	PUT
DELETE	DELETE

OBTENCIÓN DE RECURSOS

- Supongamos que tenemos las siguientes representaciones XML

URI:
http://<domain>/students/Jane

```
<student>
    <name>Jane</name>
    <age>10</age>
    <link>/students/Jane</link>
</student>
```

- URI:
http://<domain>/students

```
<students>
  <student>
    <name>Jane</name>
    <age>10</age>
    <link>/students/Jane</link>
  </student>
  <student>
    <name>Jane</name>
    <age>10</age>
    <link>/students/Jane</link>
  </student>
</students>
```

PETICIÓN TIPO: GET/RETRIEVE

- Permite **obtener** un elemento o una lista de elementos.
- Según la URI utilizada podemos:
  - GET http://<domain>/students → listado
  - GET http://<domain>/students/resourceId → elemento
- Nos devolverá el/los recurso/s en el formato indicado en la cabecera.
- En la cabecera de respuesta además se incluirá el **código de estado**:
  - **200 → OK**
  - 304 → Not modified
  - 404 → Not found
  - 500 → Internal server error

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

PETICIÓN TIPO: POST/CREATE

- Se utiliza para **crear** un nuevo recurso.
- En la uri de la petición **NO** se indica el ID del recurso:
  `POST http://<domain>/students/`
- Se ha de añadir el contenido del recurso como parámetros, ejemplo:

```
<student>
   <name>Ricardo</name>
   <age>10</age>
   <link></link>
</student>
```

- En este caso el campo `<link>` se completa en servidor.
- El servidor responde con el código 201 indicando que se ha creado.
- Además puede incluir la URI del nuevo recurso en la cabecera:
  `Location: http://<domain>/students/Ricardo`

PETICIÓN TIPO: PUT/UPDATE

- Este método se utiliza para **actualizar** una URI.
- En caso de que no exista se **crearía**.
- En la petición se ha de indicar el identificador:

```
PUT http://<domain>/students/Roberto
```

- Para actualizar un recurso primero tendríamos que obtener la representación en el cliente, actualizarlo y volverlo a enviar por PUT con los nuevos datos.
- Como respuesta nos devolverá un código indicando que se ha modificado correctamente: 200 (ok), 201 (created).
- O un código de error si no se ha podido modificar.

PETICIÓN TIPO: DELETE/DELETE

- Permite borrar un recurso.
- En la petición se ha de indicar el ID del recurso:

```
DELETE    http://<domain>/studes/Jane
```

- Como respuesta se nos devolverá un código indicando si se ha realizado correctamente (200) o si ha habido algún error (500, etc.)

ACLARACIONES

- REST vs RESTful?
- Formato devuelto: XML vs JSON?
- Es necesario implementar todos los métodos?
- Códigos de estado devueltos?
- Resumen:

Método	URI	Descripción
GET	http://<domain>/resource	Obtener todos
GET	http://<domain>/resource/id	Obtener uno
POST	http://<domain>/resource	Crear
PUT	http://<domain>/resource/id	Actualizar
DELETE	http://<domain>/resource/id	Eliminar

¿PREGUNTAS?