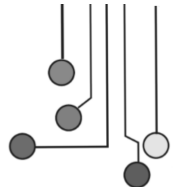


REST básico

Sistemas de Información Orientados a Servicios

RODRIGO SANTAMARÍA

- Invocación de un servicio web
  - Autenticación
  - Navegadores
  - cURL
  - Java
- Creación de un servicio web en Java

REST básico

Invocación de un servicio web

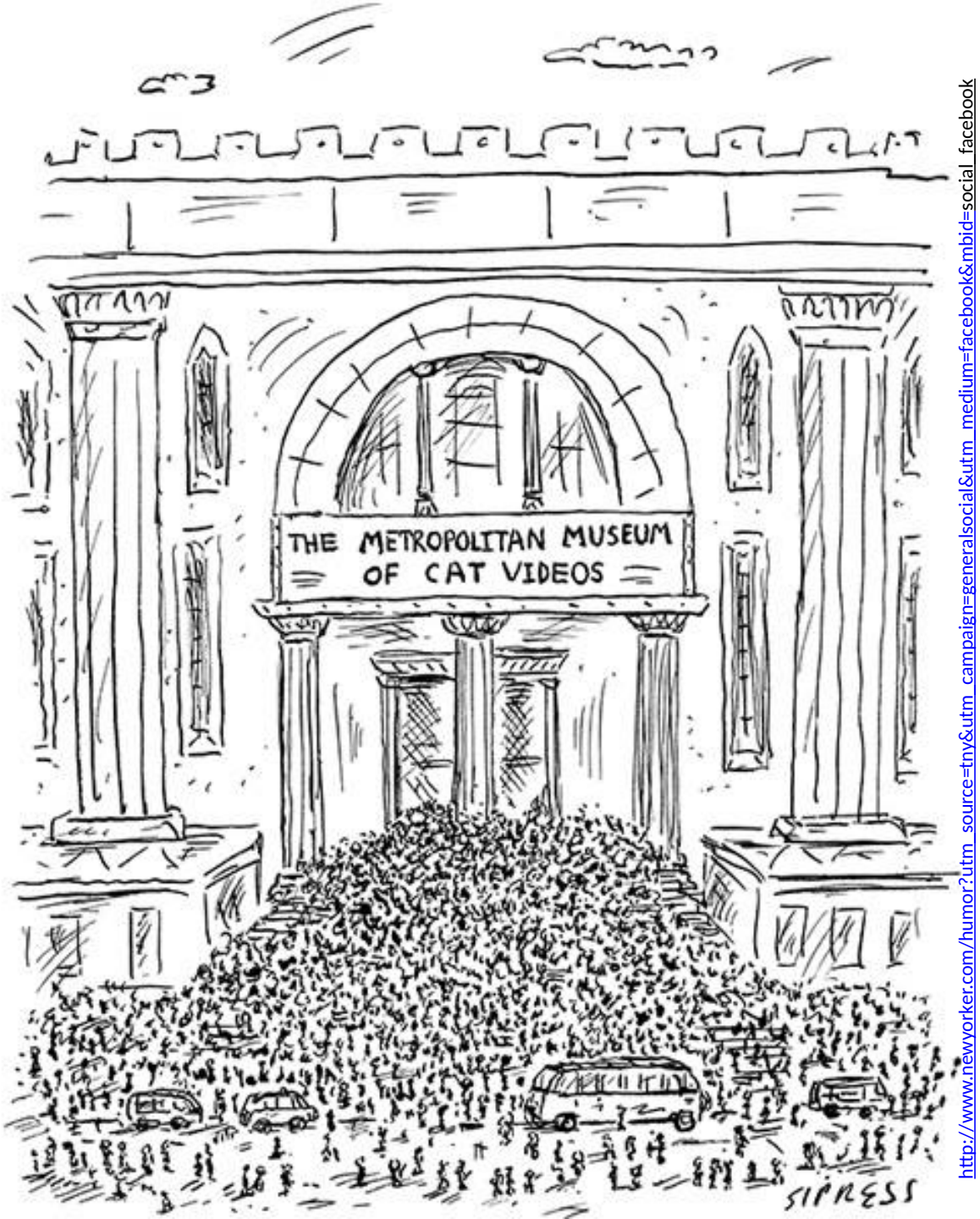
3

AUTENTICACIÓN
NAVEGADORES
CURL
JAVA

Uso de un servicio web

- Existen muchos servicios web cuya API se puede utilizar (generalmente, previa autenticación)*
- Una buena compilación: https://rapidapi.com
- Un ejemplo: https://thecatapi.com

* MacManus, R. *2021 State of APIS: Ubiquitous, Diverse, Ocasionally Open*. The New Stack, **2021** (enlace)



http://www.newyorker.com/humor?utm_source=tny&utm_campaign=generalsocial&utm_medium=facebook&mbid=social_facebook

Autenticación

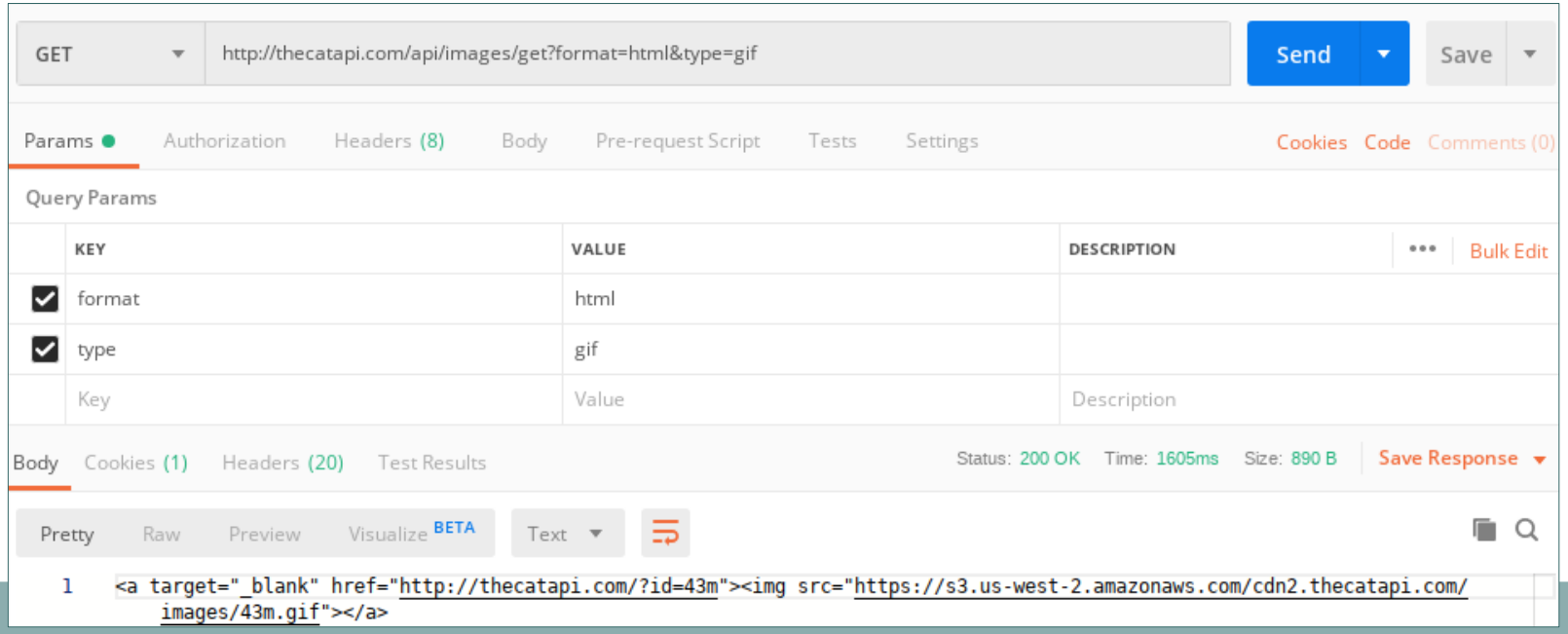
- Actualmente, casi todos los servicios web requieren algún tipo de autenticación previa
  - Generalmente a través de **OAuth** (Open Authorization), un protocolo de autenticación de APIs
    - O mediante algún sistema más sencillo de registro
  - Complica las invocaciones a la API (sobre todo de manera ‘manual’)
  - Mejora la seguridad de los servidores de servicios web

cURL

- cURL es una orden UNIX para intercambio de información mediante el protocolo HTTP
  - Es un modo rápido y efectivo de probar servicios web manualmente
- Sintaxis
  - `curl [opciones] 'url'`
  - opciones
    - `--request 'tipo'` (GET, POST, UPDATE, DELETE)
    - `--header 'cabecera'`
    - `-k` evita uso de certificados
    - `--data 'datos adicionales'`
  - ejemplo:
    - `curl 'https://api.thecatapi.com/v1/images/search'`

Postman

- Aplicación para desarrollo de APIs
  - Open source
  - Testeo similar a curl pero en GUI (requiere login)
  - https://www.getpostman.com



Java

Utilizamos clases de *java.net* y *java.io*, como para acceder a cualquier otro recurso web:

```
URL url = new URL('https://api.thecatapi.com/v1/images/search');
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.setRequestMethod("GET");

if (conn.getResponseCode() != 200) {
        throw new RuntimeException("Failed : HTTP error code : "
                        + conn.getResponseCode());
        }

BufferedReader br = new BufferedReader(new InputStreamReader(
                        (conn.getInputStream())));

String output;
                System.out.println("Output from Server .... \n");
                while ((output = br.readLine()) != null) {
                        System.out.println(output);
                }

conn.disconnect();
```

Creación de un servicio web

10

JAX-RS Y ANOTACIONES
ECLIPSE + TOMCAT + JERSEY
SERVIDOR Y CLIENTE
INTERFACES

JAX-RS

- Para crear un servicio web necesitamos algo más que los objetos de Java para manejo de conexiones
- JAX-RS (Java API for RESTful web services) es una API de Java para crear servicios web tipo REST
  - Jersey (jersey.java.net) es su implementación más estable
- Un objeto java (*POJO* – Plain Old Java Object) se convierte en un recurso web añadiéndole ***anotaciones***
  - Sintaxis incorporada a Java en la versión 1.5
  - Provee información sobre el código, pero no es código
    - Información para la compilación, desarrollo o ejecución

JAX-RS: anotaciones

- @Path indica la ruta relativa a añadir a la URI para acceder a una clase o método
- @GET, @PUT, @POST, @DELETE, @HEAD hacen referencia al tipo de petición HTTP que satisface un método
- @Produces especifica el tipo MIME que retorna (plain, html, json, xml, etc.) un método
  - @Consumes especifica el tipo MIME que requiere un método
- Existen más anotaciones, éstas son sólo las esenciales

JAX-RS: anotaciones

- Por ejemplo:

```
@GET
@Produces(MediaType.TEXT_PLAIN)
@Path("/saludo");
public String saludar(){ return "Hola"; }
```

- Retornará un mensaje en texto plano que dice "Hola" al acceder a http://host:port/saludo (método GET)

Esquema

JAX-RS

Anotaciones Java 1.5

Cliente REST (vía JAX-RS)

```
ClientConfig conf = new DefaultClientConfig();
Client client = Client.create(conf);

URI uri=UriBuilder
      .fromUri("http://ip:8080/servicePath").build();
WebResource service= client.resource(uri);

System.out.println(service.path("classPath")
      .path("hello").accept(MediaType.TEXT_PLAIN)
      .get(String.class))
```

Servicio REST

```
@GET
@Produces(MediaType.TEXT_PLAIN)
@Path("hello");
public String saludar(){ return "Hola"; }
```

Preparación del entorno

- Descargar **Tomcat** 6.0 de http://tomcat.apache.org/
- Descargar **Eclipse**
  - Instalar plugin para desarrollo web: WTP
    - Help/Install New Software...
    - http://download.eclipse.org/releases/indigo
    - Web, XML, Java EE, etc.
  - O bien descargar la versión para desarrolladores EE
- Descargar **Jersey** (http://jersey.java.net), buscar el enlace en Downloads (JAX-RS 2.0 API jar)
  - Al crear el proyecto tendremos que agregar dichos jars

Creación del proyecto

- Crear un nuevo proyecto web:
  - File/New/Project… → Web/Dynamic Web Project
- En la carpeta `WebContent/WEB-INF/lib`, incluir todos los jars que hay en las carpetas `jersey/lib`, `jersey/api` y `jersey/ext`

- *En http://vis.usal.es/rodrigo/documentos/sisdis/ejemploREST/ se encuentran algunas de las clases y ficheros que vamos a usar de ejemplo*

Creación de un servicio REST
Fichero web.xml

- Modificar el fichero `WebContent/WEB-INF/web.xml` por este otro:
  - http://vis.usal.es/rodrigo/documentos/sisdis/ejemploREST/web.xml
- `display-name` debe coincidir con el nombre del proyecto
- `jersey.config.server.provider.packages` debe tener como valor una lista de nombres de paquetes en los que tenemos recursos REST, separados por punto y coma.
- `url-pattern` dentro de `servlet-mapping` debe ser la ruta base a partir de la que se ubicarán los recursos REST

Creación de un servicio REST
Ejemplo de servicio


```
//Sets the path to base URL + /hello
@Path("/hello")
public class Hello
        {
        // This method is called if TEXT_PLAIN is request
        @GET
        @Produces(MediaType.TEXT_PLAIN)
        public String sayPlainTextHello() {
          return "Hello Jersey";
          }

        // This method is called if XML is request
        @GET
        @Produces(MediaType.TEXT_XML)
        public String sayXMLHello() {
          return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
          }
        // This method is called if HTML is request
        @GET
        @Produces(MediaType.TEXT_HTML)
        public String sayHtmlHello() {
          return "<html> " + "<title>" + "Hello Jersey" + "</title>"
             + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
          }
        }
```

Creación de un servicio REST
Ruta del servicio

- http://ip:8080/proyecto/servlet/clase/metodo

localhost o la ip del equipo remoto
(mejor ips que nombres, pues pueden estar corruptos en el lab. de informática)

Indicado con la anotación @Path antes de un método.
Puede no usarse si se tiene ruta ya en la clase

Indicado con la anotación @Path antes de una clase.
Podemos no usarlo, pero es recomendable para guardar un cierto orden

Indicado en el tag <url-pattern> de <servlet-mapping> en web.xml
p. ej. si queremos que sea servlet usamos /servlet/*
Podemos no usarlo, poniendo simplemente /*

nombre del proyecto en el IDE, que debe coincidir con el tag <display-name> de web.xml

Creación de un servicio REST
Arranque del servicio

- Arrancar el servicio: Run/Run As…/Run on Server
  - Especificar Tomcat como servidor en el que arrancarlo
    - Target runtime (o *New…* si no está)
- Errores frecuentes:
  - **java.lang.ClassNotFoundException**: com.sun.jersey.spi.container.servlet.ServletContainer
    - Los jar de Jersey no se han incluido correctamente en WebContent/WEB-INF/lib
  - **com.sun.jersey.api.container.ContainerException**: The ResourceConfig instance does not contain any root resource classes.
    - El parámetro com.sun.jersey.config.property.packages no se ha configurado correctamente en web.xml: debe contener los nombres de los paquetes que contienen clases anotadas.
  - El servidor arranca pero no hay nada en las rutas esperadas
    - El parámetro com.sun.jersey.config.property.packages no se ha configurado correctamente en web.xml: debe contener los nombres de los paquetes que contienen clases anotadas.
    - Revisar los @Path, y los tags <display-name> y<servlet-mapping> en web.xml

Creación de un servicio REST
Ejemplo de cliente


```
public class Test {
        public static void main(String args[])
                {
                Client client=ClientBuilder.newClient();;
           URI uri=UriBuilder.fromUri("http://localhost:8080/pruebasREST").build();

           WebTarget target = client.target(uri);

                System.out.println(target.path("rest").path("hello").request(MediaType.TEXT_PLAIN).get
        (String.class));
                System.out.println(target.path("rest").path("hello").request(MediaType.TEXT_XML).get
        (String.class));
                System.out.println(target.path("rest").path("hello").request(MediaType.TEXT_HTML).get
        (String.class));
                }
        }
```

Se ejecuta como una aplicación Java normal

Ejercicio

- Crear un servicio REST *hello* mediante Eclipse, Tomcat y Jersey.
  - Iniciar en la máquina local y probar accesos de clientes
    - Desde un navegador y desde java
    - Desde la máquina local y desde otras máquinas

Creación de un servicio REST
Paso de argumentos

- Paso de argumentos: anotación **@QueryParam**:

```
@Path(”calculator")
public class Calculator
  {
  @Path(“sq")
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  public String square(@DefaultValue("2") @QueryParam(value="num") long num)
    {
    return ""+num*num;
    }
  }
```

- Desde URL http://hostname:port/calculator/sq?num=3
- Desde Java
  - service.path("calculator/sq").queryParam("num", ””+3).request (MEDIATYPE.TEXT_PLAIN).GET(STRING.**CLASS**)

Creación de un servicio REST
Retorno de objetos

- En principio, Jersey retorna tipos MIME (es decir, texto, en distintos formatos)
  - Jersey no soporta la serialización de tipos primitivos
  - Debemos usar String + documentación de la API
  - Si intentamos, por ejemplo, retornar un long:
    - com.sun.jersey.api.MessageException: A message body writer for Java class java.lang.Long, and Java type long, and MIME media type XXX was not found
- Solución: convertir objetos Java en texto (p. ej. XML)
  - Jersey da soporte para ello a **JAXB**, una arquitectura para asociar clases Java a representaciones XML

Creación de un servicio REST
Minimización de interfaces

- Respecto al uso de argumentos y el retorno de objetos, un buen diseño de un sistema distribuido minimiza las interfaces
  - Suponen una carga en el tráfico de red
    - Y más si hay que convertirlos a XML
  - Incrementan el riesgo de errores
    - Interpretaciones equivocadas de la API
    - Las clases tienen que estar disponibles por los clientes
    - Etc.
  - Muchos objetos son evitables con un uso inteligente de String

Creación de un servicio REST

Ciclo de vida de los objetos

- En Jersey, los objetos tienen un ciclo de vida '*per-request*'
  - Cada clase que se ofrece como recurso se instancia con cada nueva petición y se destruye al terminar dicha petición
    - Esto impide mantener objetos que varían su estado a lo largo del tiempo (a través de distintas peticiones)
  - Solución:
    - Utilizar la anotación @**Singleton** para la clase
      - Así, la clase se instancia una vez por aplicación web, y permanece instanciada hasta que se apague o reinicie el servicio

Ejercicio

- Crear un servicio REST *calculator* que permita realizar potencias cuadradas (*sq*) y sumas de dos elementos (*add*)
  - Obtendrá mediante parámetros el número a elevar al cuadrado y los dos números a sumar (todos enteros)
  - Retornará el resultado como una cadena de texto

- Añadir una tercera función *stack*(int *n*) que sume el valor *n* a una variable interna del servicio que comienza en *0*

Tutoriales

- http://www.vogella.com/articles/REST/article.html
  - Preparación básica para trabajar con Jersey+Tomcat+Eclipse
- https://jersey.java.net/documentation/latest/user-guide.html
  - Manual completo de Jersey, en especial:
    - Paso de argumentos (cap 3.2)
    - Ciclo de vida de los recursos (3.4)