

# DEW 2022/23

## TRABAJO EN GRUPO NOTAS ONLINE

PROF. RAMÓN GARCÍA

### TABLA DE CONTENIDO

1	En resumen.....	4
2	Organizando las piezas .....	5
3	El problema de las dependencias.....	7
4	Practicando el desarrollo de piezas individuales .....	8
5	Anotaciones (log) sobre accesos .....	9
6	CentroEducativo: el nivel de datos .....	9
6.1	Trabajando con CentroEducativo .....	10
7	Informaciones y Herramientas HTTP y REST .....	10
7.1	Interacciones con un servidor REST en general .....	11
7.2	Inspección de detalles de CentroEducativo mediante REST .....	13
7.3	Invocación con detalles HTTP (curl) .....	16
7.4	Invocación por programa (en Java).....	18
7.4.1	Cómo construir una petición HTTP y recibir la respuesta .....	18
7.4.2	Cómo codificar y decodificar datos en JSON .....	18
8	Gestión de la identidad (autenticación) .....	19
8.1	Configuración de Tomcat .....	19
8.2	Compatibilizando sesiones web y de datos .....	20
9	La vista del usuario: navegación, contenidos, aspecto .....	22
9.1	La página de bienvenida/login .....	22
9.2	Las páginas de navegación intermedias .....	23
9.2.1	Las páginas múltiples .....	23
9.3	Las páginas finales de detalle.....	24
9.3.1	El caso especial de la operación AJAX .....	24
10	Primeras operaciones sobre la aplicación.....	26
10.1	Usuarios del rol pro .....	26
10.2	Usuarios del rol alu.....	26
10.3	Asignaturas.....	27
10.4	Caso a resolver .....	27

11	Selección de operaciones .....	28
12	AJAX: Más piezas a desarrollar .....	31
13	Organización y transferencia de las imágenes .....	32
14	Evaluación .....	33
14.1	Hitos y entrega final .....	33
14.1.1	Entrega final .....	33
14.2	Prueba individual (selectiva) corresponsable .....	34
14.3	Coevaluación .....	35
15	Bibliografía .....	35
16	Anexo: Servlets actuando como clientes HTTP. Operativa básica y gestión de cookies. 36	

El trabajo propuesto consiste en el desarrollo de una aplicación web (no12223) basada en un blog, con Bootstrap, jQuery, servlets, e interacción por AJAX. El resultado deberá incluir una memoria descriptiva, el código fuente debidamente documentado, y la aplicación instalada en la máquina de portal correspondiente.

Lunes (3Tl2l)	24/04	08/05	15/05	22/05	29/05	05/06
Miércoles (2*3Tl1x)	26/04	03/05	10/05	17/05	24/05	31/05

Tal y como se ha mencionado a lo largo del tema sobre servlets, usaremos Java 11 y la API 2.5 para servlets en un contenedor Tomcat 9.

### RELACIÓN ENTRE SESIONES (HITOS) Y APARTADOS DE ESTE DOCUMENTO

Sesión	Apartados
<b>0</b>	1, 2, 3, 4
<b>1</b>	2, 5, 6, 7.1, 7.2, 7.3
<b>2</b>	2, 7.4, 8, 10.1, 10.2, 10.3, 12 (solo URL)
<b>3</b>	2, 9, 10.4, 11, 16
<b>4</b>	2, 12, 13
<b>5</b>	2, 12, 13

Los apartados no citados mencionan aspectos no relacionados con sesiones específicas (forma de calificación, bibliografía y otros)

## 1 EN RESUMEN...

Se trata de la implantación en un centro educativo de un sistema (**NOL** – *Notas On-Line*) que permita gestionar las calificaciones de los alumnos. Para ello aparecen dos tipos de usuarios (alumnos y profesores) con capacidades y responsabilidades bien definidas. Los objetos que se definen (asignaturas) son una gran reducción de la realidad.

Resumidamente, los *alumnos* usan el sistema **NOL** para averiguar sus calificaciones en las pruebas de las asignaturas en las que se encuentren matriculados. No pueden consultar calificaciones de otros alumnos. Los *profesores* usan **NOL** para crear las pruebas y asignar puntuaciones para cada alumno matriculado en sus asignaturas (reducimos el caso a un profesor por asignatura, pero varias asignaturas por profesor).

Para aumentar su relación con las tecnologías de la web, algunas características a exigir serán:

- Acabado uniforme de todas las páginas, basado en Bootstrap. Hay que cuidar especialmente el sistema de navegación, que debe ser coherente.
- Añadir *logs*<sup>1</sup> como filtro controlable (que pueda activarse o no)
- En algunas operaciones se solicitará una versión AJAX
- Autenticación básica (Basic) (se necesitará completarlo programáticamente)
- Las calificaciones de un alumno pueden reunirse en un certificado de notas, especialmente preparado para su impresión en papel.

---

<sup>1</sup> Archivo/s de anotaciones cronológicas que reflejen las operaciones realizadas

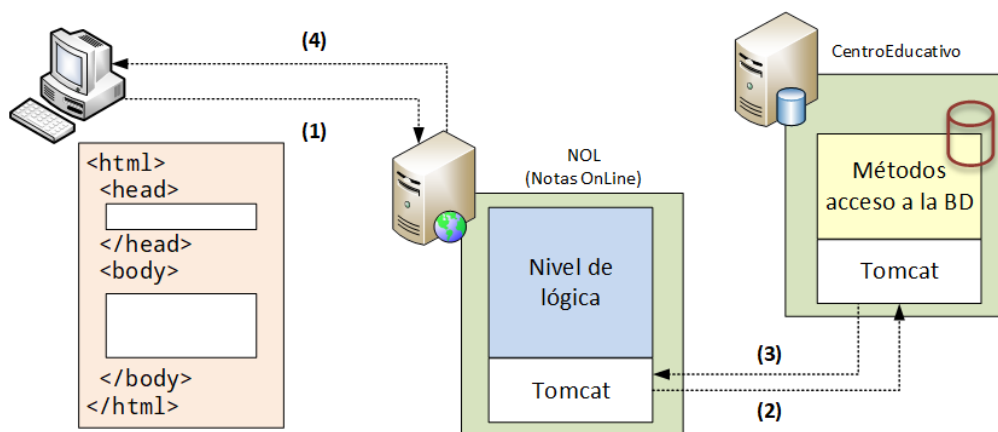
## 2 ORGANIZANDO LAS PIEZAS

Tenemos una aplicación en la que interactúan 3 niveles:

1. El nivel de interfaz de usuario se implementa mediante las páginas de hipertexto devueltas por el servidor web y mostradas por el navegador del cliente.
2. El nivel de aplicación (NOL, en nuestro caso) contiene la lógica para atender las solicitudes interactuando con la información disponible.
3. El nivel de datos (CentroEducativo, en nuestro caso) mantiene los datos persistentes que ofrece al nivel de aplicación.

Dado que se ha optado por un modelo REST para el nivel de datos, el único protocolo de comunicación existente entre estos niveles será HTTP. Esto permitiría optar, al menos, por un par de organizaciones interesantes:

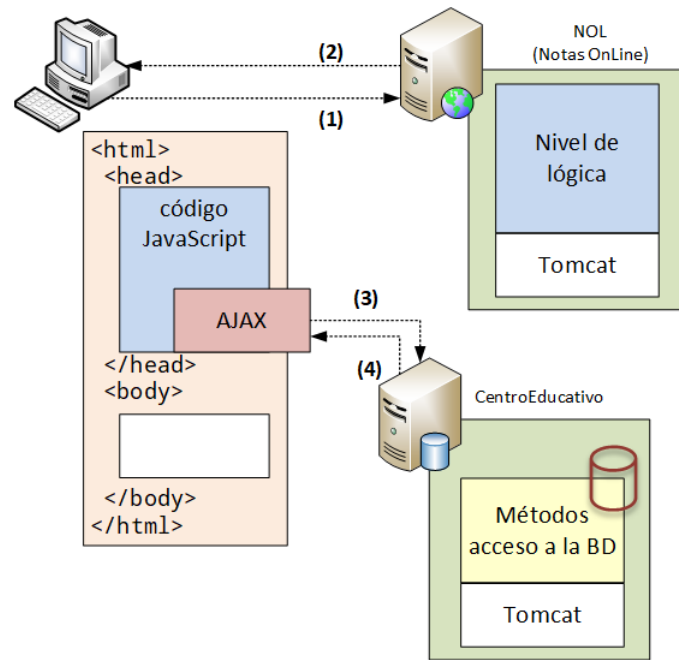
1. El nivel de lógica oculta al cliente la existencia del nivel de datos. Para cada petición que deba atender, la lógica interactuará mediante el API REST con los datos, construirá la página de resultados y la devolverá al cliente solicitante. En el cliente no se precisa el uso de AJAX puesto que solo interactúa con el nivel de lógica



(la página HTML se obtiene en el paso 4)

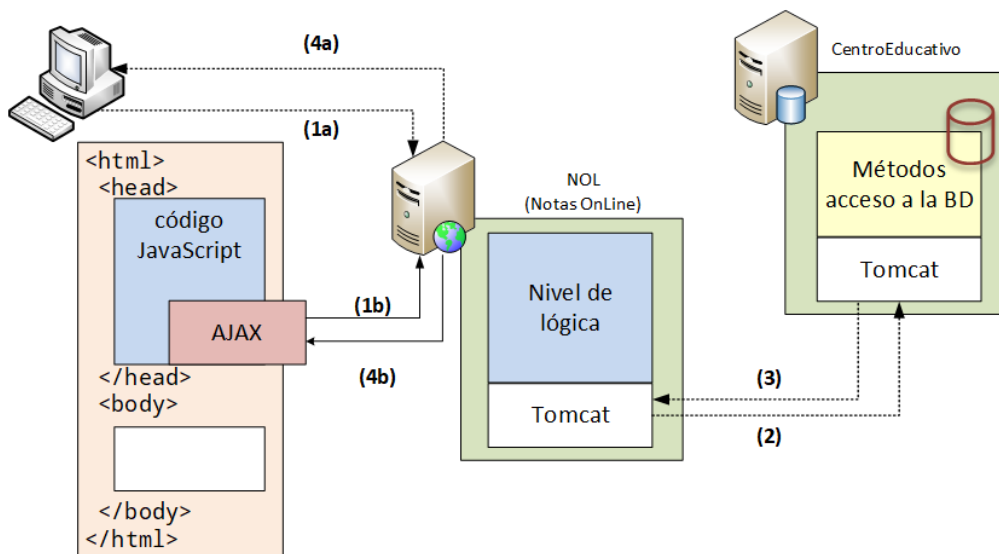
- Si se desea evitar el patrón *Cliente -> Servlet -> API REST*, se debería añadir la flexibilidad necesaria para que los diferentes servicios elijan la forma de ofrecer sus respuestas (HTML, JSON, ...) dependiendo del tipo de cliente.
2. El usuario accede a la API REST del nivel de datos mediante solicitudes AJAX. En este caso, el nivel de lógica devuelve al navegador una página web compleja<sup>2</sup>, que accederá a la información necesaria mediante AJAX que aproveche la API REST. En esta aproximación el navegador corre con la parte de procesamiento más importante (es un cliente *pesado*).
    - Se pueden destacar un par de problemas en esta aproximación: la dificultad para resolver algunos aspectos de seguridad, y el rendimiento de la parte cliente de la aplicación, que puede ser insuficiente.

<sup>2</sup> Tal y como se define en el apartado 2



(la página HTML se obtiene en el paso 2)

Elegimos la primera aproximación (el cliente no accede directamente al nivel de datos), pero permitimos interacciones AJAX entre cliente y NOL en algunos casos que elegiremos. Esto nos lleva a este tercer y último esquema:



(El esquema representa 2 circuitos para 2 tipos diferentes de cliente: la secuencia iniciada por el usuario 1a-2-3-4a, y la secuencia iniciada por una petición AJAX 1b-2-3-4b)

Pueden darse varios planteamientos para ofrecer una API diferenciada a los dos posibles tipos de cliente, puesto que en las respuestas 4a se devuelven objetos MIME completos, y en las 4b se devuelven objetos JSON arbitrarios. El profesor no va a formular propuestas concretas sobre este tema.

### 3 EL PROBLEMA DE LAS DEPENDENCIAS

En muchas ocasiones se consideran inamovibles aspectos o configuraciones que en realidad no lo son. Citamos como ilustración un par de escenarios que todos conocemos:

- Se realiza un desarrollo y se prueba en casa. Colocando el mismo desarrollo en otro equipo, especialmente en un equipo de la universidad, la aplicación no funciona como debiera.
- Se realiza un desarrollo y se prueba en el equipo de la universidad de un compañero. Colocando el mismo desarrollo en otro equipo o cuenta, no funciona como se espera.

La explicación de estos errores, sin entrar en detalles, es que pensamos que las condiciones bajo las que la aplicación funcionaba correctamente también se iban a dar en el caso en el que finalmente falló. Entramos ahora en algunos de esos detalles:

- El sistema de ficheros es diferente: la ruta a los **archivos**, sus permisos y propietarios, el directorio actual cuando la aplicación se ejecuta, las diferencias enormes entre LINUX y Windows, ... pueden ser motivos de fallo. En ocasiones la solución es sencilla: colocar la aplicación y los archivos que maneja en un directorio controlado.
- En nuestra lista de dependencias no hemos contado con que las **bibliotecas** pueden no existir o ser diferentes. En algunos casos se requerirá una acción extra, pero a veces no es un problema con solución y debería ser conocido cuanto antes.
- En nuestra lista de tecnologías, productos y bibliotecas no hemos acertado controlando las **versiones** que se encuentran instaladas en las diferentes máquinas. Las aplicaciones Java, por ejemplo, tienen problemas propios cuando las versiones de las máquinas virtuales (MVJ) cambian.

Una aplicación correcta para una versión A de la MVJ puede tener problemas al ejecutarse sobre otra versión B:

- Si  $A > B$ , la MVJ B puede no disponer de la funcionalidad propia (API) de A
- Si  $B > A$ , la MVJ B no dispondrá de las funciones que en A se hayan marcado como *deprecated*
- A veces colocamos de forma inflexible informaciones que deberán ser **adaptadas** a otros casos. Imaginad que se necesita una acreditación (*login* y *clave*) para efectuar una conexión con un SGBD desde una aplicación web en Java. No sería extraño que esa operación aparezca en el código de todos los servlets, por lo que su modificación es tediosa y, además, se requiere recompilación. Por otro lado, el propio SGBD puede cambiar de ubicación (puerto e IP), también pondrá a prueba que hayamos previsto esa posibilidad.

En las aplicaciones web en Java, el descriptor de despliegue `web.xml` es el lugar natural para anotar esas informaciones en parámetros de contexto, de manera que en ejecución pueden ser consultadas con operaciones como `getServletContext().getParameter("nombre_dependencia")`.

En ABSOLUTAMENTE todos los casos es **necesaria una etapa de comprobación** anterior a la entrega final de la aplicación. En ella se debe poner a prueba ese desarrollo, comprobando que

realiza lo esperado, y que la seguridad no se rompa. No es extraño apurar los plazos finales y que esta última comprobación se efectúe deficientemente. Debe ser incluida en nuestras previsiones como una etapa de riesgo<sup>3</sup>.

#### 4 PRACTICANDO EL DESARROLLO DE PIEZAS INDIVIDUALES

Tras las consideraciones anteriores, descriptivas y generalistas, pasamos ahora a la realización de actividades que vayan resolviendo gradualmente aspectos parciales de interés en nuestro caso. Para practicar, realiza estas actividades:

1. Crear una página simple completa
2. Crear un servlet que devuelva una página simple completa. Tres opciones:
  - Redirección
  - Mediante RequestDispatcher
  - Construyéndola con órdenes `out.print()`

La última alternativa es la más flexible, pero engorrosa

3. *(Solo si en el temario ya has llegado a la parte de AJAX)* Crear un servlet que devuelva un objeto JSON (sin necesidad de biblioteca). Puedes emplear el que se muestra en el próximo punto, pero sin AJAX no podremos "consumir" el resultado devuelto

Definimos como **página compleja** aquella que modifica dinámicamente su estructura y/o contenido mediante Javascript (y jQuery). En este momento hay un vacío entre los puntos 3 y 4, porque somos capaces de producir y de consumir, pero no podemos conectar al que produce con el que consume (necesitamos AJAX para ello).

4. Crear una página compleja con *datos locales*. Representa internamente esos datos mediante un objeto JSON que los contenga. Generaliza para el caso en que el objeto contenga un array como una de sus propiedades.

```
var objetoJSON = {nombre: "Es un ejemplo",
  apellidos: "Mis apellidos",
  diasPreferidos: ["lunes", "miercoles", "jueves"]}
}
```

Crear una página que emplee ese objeto JSON (es un dato local) para colocarlo como parte del contenido. La forma de usarlo será: por cada elemento del vector, hay que añadir un párrafo que incluya el día, los apellidos y la longitud del nombre.

5. Crear un servlet que devuelva una página compleja completa (la usada en dew3: [http://personales.upv.es/rgarcia/dew3/quijotoc\\_tdc\\_listaNúm\\_parpadeo.html](http://personales.upv.es/rgarcia/dew3/quijotoc_tdc_listaNúm_parpadeo.html)).
6. Modifica el servlet anterior para que sustituya el primer enlace (*wikisource*) por otro diferente en cada invocación del servlet (p.ej. al pronóstico del tiempo, a El País, a la UPV,... algo diferente cada vez)

<sup>3</sup> En la acepción de la dificultad para delimitar su duración



## 5 ANOTACIONES (LOG) SOBRE ACCESOS

Debe desarrollarse un filtro de entrada (logs) que anote en un fichero *persistente* los accesos y momentos de activación de los diferentes servlets. P.ej. si el día 9 de junio de 2020 a las 19:38:14.278, el usuario prof1 desde la máquina 158.11.11.11 ejecuta el servlet acceso, por el método GET, entonces en el fichero persistente debe aparecer una entrada como ésta:

```
2020-06-09T19:38:14.278 prof1 158.11.11.11 acceso GET
```

**Nota.** Te sugerimos usar la función `LocalDateTime.now()` de `java.time.LocalDateTime`. La ubicación del archivo persistente debe establecerse en `web.xml`, y su contenido debe permanecer ordenado cronológicamente.

El planteamiento es incremental, de manera que se pretende crear:

1. Una versión 0 que devuelve texto con informaciones relevantes (datos del formulario, información del cliente, fecha actual, URI, método)
2. Una versión 1 que almacena una copia del resultado en un fichero con nombre y ruta preestablecidos. Se debe estudiar el problema de acceso a ficheros locales fuera del espacio web del servidor, que tiene implicaciones de cierta importancia.
3. En la segunda versión debe consultarse en `web.xml` algún dato que especifique la ruta del archivo.
4. La conversión del servlet `logs` para que actúe como filtro se realizará posteriormente, activándose al acceder a cualquier página o servlet de la aplicación

Debe crearse un formulario para poner a prueba cada versión.

## 6 CENTROEDUCATIVO: EL NIVEL DE DATOS

Como parece natural un problema como éste tiene implicaciones en el ámbito de los sistemas de información: es importante reflexionar acerca de las informaciones que se emplean, cómo se van a usar y cuál es la mejor organización aplicable. Por ello el estudio de estos aspectos debería ser parte de este trabajo, con la salvedad de que... *¡no es parte de los objetivos de DEW!*. Por ello no planteamos cómo modelar el problema, ni cómo diseñar la organización para poder apoyarnos en un gestor de bases de datos como herramienta más adecuada (que lo es).

Para independizarnos parcialmente del nivel de persistencia de datos, sería aconsejable establecer una interfaz REST, actuando como contrato entre nuestro nivel de lógica y el de datos. Por ello se ha realizado<sup>4</sup> una aplicación (CentroEducativo) que encapsula los datos y el acceso a los mismos, y puedes descargar desde la carpeta Labo2-NOL del área de recursos de PoliformaT (`es.upv.etsinf.ti.centroeducativo-0.2.0.jar`).

Este software ha dejado de estar sincronizado con las versiones actuales de Java, debido a una biblioteca no soportada. Resolvemos el problema usando Java 8, para este caso exclusivamente:

```
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java -jar es.upv.etsinf.ti.centroeducativo-0.2.0.jar
```

<sup>4</sup> Gracias al profesor Pedro Valderas

CentroEducativo es una aplicación encapsulada que pone en marcha un servidor que atiende peticiones por el puerto 9090. En su uso final está prohibido el acceso a esta aplicación desde otra procedencia diferente a la aplicación web que desarrollamos, pero es probable que durante el desarrollo nos interese aliviar esta limitación, siendo necesario abrir el cortafuegos con:

```
ufw allow 9090/tcp
```

## 6.1 Trabajando con CentroEducativo

El procedimiento para trabajar con la aplicación consta de los siguientes pasos, destacando los detalles importantes:

0. **Situación inicial:** CentroEducativo se ejecuta sobre un equipo que atiende peticiones por el puerto 9090
1. **Primer paso: identificación.** El cliente de la aplicación realiza una operación login.
  - Aporta un identificador (dni) y una contraseña (password).
  - Como resultado recibe una clave de sesión que debe proporcionar en cada una de las operaciones siguientes (key=...).
  - Hay un diálogo interno en este primer paso, y en él se creará una *cookie* JSESSIONID. Su funcionamiento es automático en herramientas de alto nivel, pero **no** lo es en el resto de casos. Observa los detalles sobre `curl` en el apartado "Herramientas".
2. **Resto de pasos: operaciones** de consulta y modificación.
  - Ya no se necesitan los datos de identificación, pero sí la clave de sesión.
    - Estas operaciones solo funcionarán si esa clave *encaja* con la *cookie* que CentroEducativo estableció en el paso login.
  - Todas ellas devuelven un código de resultado.
  - Las de consulta devuelven además objetos JSON que deben ser recuperados y procesados.

Es muy importante destacar que **la aplicación no implementa persistencia**, por lo que su reinicio provoca la pérdida de las modificaciones realizadas y el retorno a la situación inicial. En ese sentido cobra importancia la automatización de operaciones, por ejemplo para añadir 10 alumnos y calificarlos en 3 asignaturas. El apartado relacionado con `curl` puede ser de utilidad.

## 7 INFORMACIONES Y HERRAMIENTAS HTTP Y REST

La asignatura DEW no incluye entre sus objetivos el estudio de las interfaces de aplicaciones para web, o de su arquitectura, o de los posibles modelos y herramientas. Eso no significa que no existan, simplemente no caben en nuestro presupuesto.

Por ello solamente puedo alcanzar a relacionar fuentes de información y herramientas de interés para interactuar con aplicaciones REST como "CentroEducativo". La idea es consumir operaciones accesibles mediante una API basada en REST. No deseamos ofrecer esas operaciones al usuario final (¡es una persona!), aunque nos puede interesar invocarlas desde nuestros scripts (¡son aplicaciones!).

Muy resumidamente, y especialmente para nuestro caso, en REST se aplican operaciones sobre datos colectivos o individuales siguiendo el siguiente criterio SIMPLIFICADO:

	<i>/recursos (colección)</i>	<i>/recursos/item03 (elemento)</i>
GET	Devuelve la lista de los URIs de la colección recursos	Devuelve item03
PUT		Sustituye el contenido completo de item03
POST	Crea una nueva pieza de la colección recursos y devuelve el URI que se le ha asociado	
DELETE		Elimina el elemento item03
PATCH		Actualiza parcialmente item03

Nota: PATCH no suele formar parte de los métodos mencionados en las aplicaciones REST básicas.

Los URIs mencionan objetos, y los métodos HTTP representan las posibles operaciones. Es frecuente representar una jerarquía de objetos, como alumnos de una asignatura; así, para "desmatricular" al alumno alu7 de la asignatura asi4, se podría invocar:

DELETE /asignaturas/asi4/alumnos/alu7

## 7.1 Interacciones con un servidor REST en general

Entre los posibles clientes de servicios REST destaco la extensión **RESTED** para Firefox (quizás exista para otros navegadores). Mediante ella se puede interactuar con una API REST, seleccionando método, contenido de cabeceras, etc... Como si pudieras modificar directamente un mensaje HTTP e interpretar la respuesta.

Para instalar extensiones o complementos de Firefox, se debe utilizar la opción "Complementos" del menú "Herramientas". Cuando **RESTED** se haya instalado, aparecerá un botón `</>` en la parte derecha de la barra de navegación. Basta con pulsar ese botón para abrir una pestaña **RESTED**.

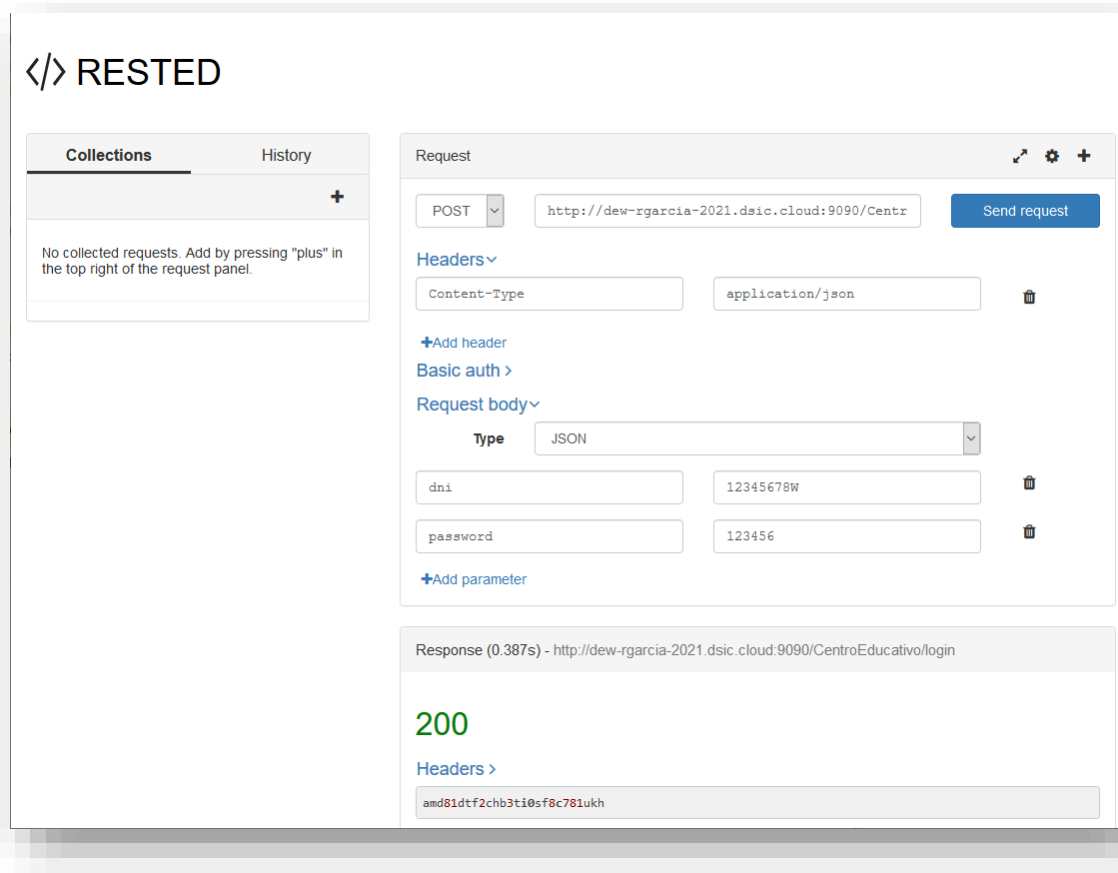
- Otra alternativa, mejor documentada, es la extensión "REST Client" (<https://marketplace.visualstudio.com/items?itemName=humao.rest-client>) para Visual Studio Code

A continuación se ilustran<sup>5</sup> dos ejemplos de uso de RESTED en la aplicación CentroEducativo.

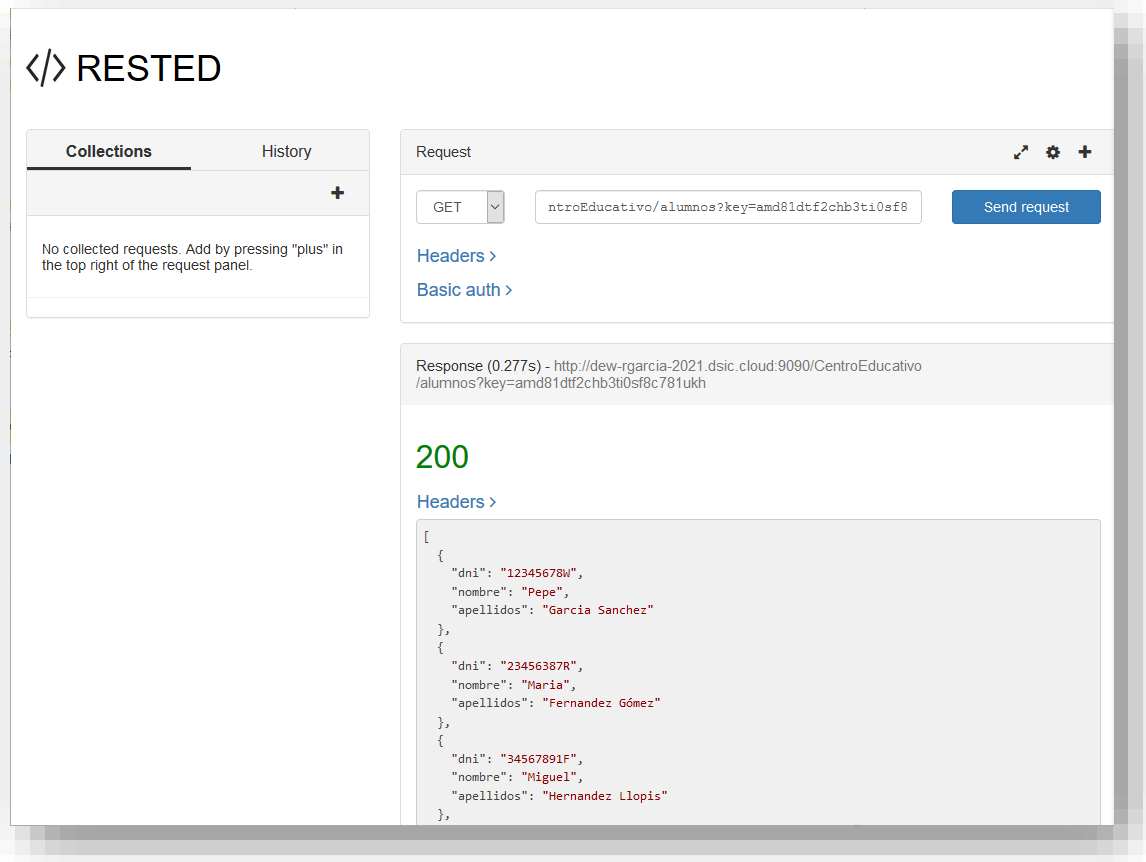
En el **primer** ejemplo se invoca la operación "login", por POST, codificando en el cuerpo del mensaje un objeto JSON con el DNI y clave de un alumno. La respuesta que recibimos es una confirmación (código HTTP 200) junto con un resultado (amd81dt...). Ese resultado es un *token* que debe ser usado a lo largo de esta sesión, como demostración de que se trata de un usuario ya identificado.

<sup>5</sup> Se han mantenido las capturas del curso pasado

- Se colocará (en otra operación) un parámetro key con la clave de sesión recibida



En este **segundo** ejemplo, un profesor ya identificado (con key= amd81dt...) invoca por GET la operación para obtener la lista de alumnos. En este caso no hay argumentos adicionales.



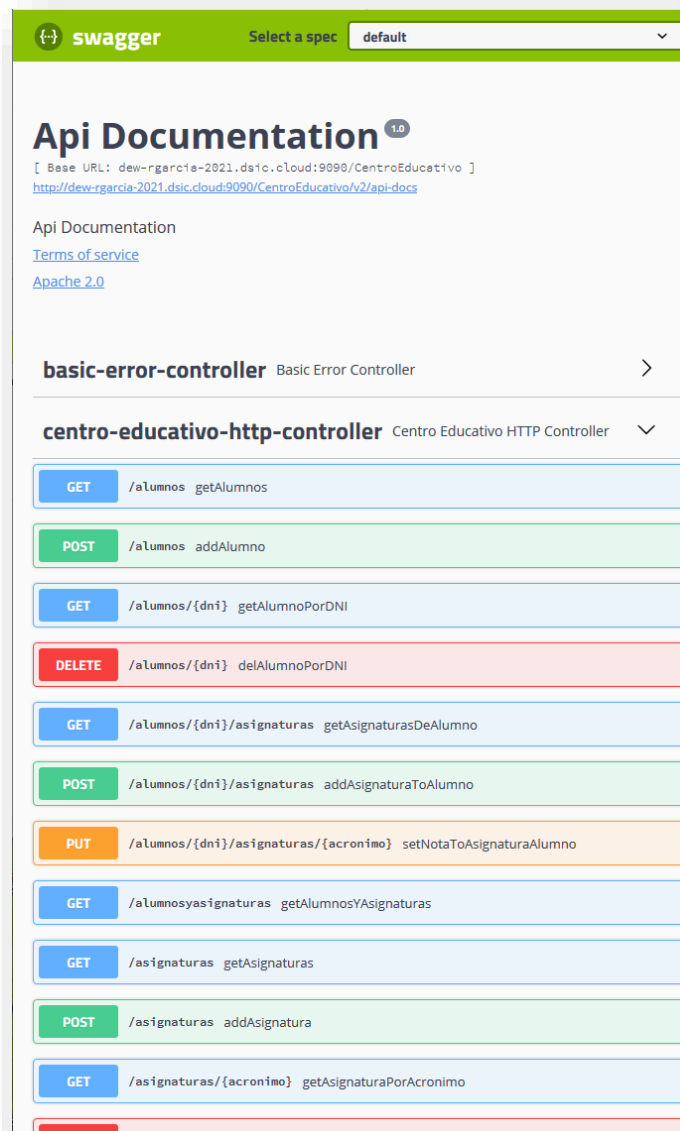
- Observa que la respuesta es un vector JSON en el que aparece una entrada por cada alumno encontrado.

Hay múltiples servicios exteriores modelados mediante REST que son consumidos desde otras aplicaciones, pero no todos permiten un uso gratuito. También los hay que requieren un registro previo (*darse de alta*) para poder operar, seguramente para evitar usos abusivos. Puede encontrar una relación de casos en la bibliografía.

## 7.2 Inspección de detalles de CentroEducativo mediante REST

Toda la documentación de la API está accesible en la propia aplicación<sup>6</sup>, en la página /CentroEducativo/swagger-ui.html con muchísimos niveles seleccionables de detalle

<sup>6</sup> Muy relacionado con la especificación OpenAPI



The image shows a Swagger UI interface for an API. At the top, there's a green header with the Swagger logo and a dropdown menu labeled "Select a spec" with "default" selected. Below the header, the title "Api Documentation" is displayed with a version number "1.0". Underneath the title, there's a base URL: "[ Base URL: dew-rgarcia-2021.dsic.cloud:9090/CentroEducativo ]" and a link to the API docs: "http://dew-rgarcia-2021.dsic.cloud:9090/CentroEducativo/v2/api-docs".

Below the title, there's a section for "Api Documentation" with links for "Terms of service" and "Apache 2.0".

The main content area is divided into two sections. The first section is "basic-error-controller" with a description "Basic Error Controller" and a right arrow. The second section is "centro-educativo-http-controller" with a description "Centro Educativo HTTP Controller" and a dropdown arrow. This section contains a list of API endpoints:

- GET /alumnos getAlumnos
- POST /alumnos addAlumno
- GET /alumnos/{dni} getAlumnoPorDNI
- DELETE /alumnos/{dni} delAlumnoPorDNI
- GET /alumnos/{dni}/asignaturas getAsignaturasDeAlumno
- POST /alumnos/{dni}/asignaturas addAsignaturaToAlumno
- PUT /alumnos/{dni}/asignaturas/{acronimo} setNotaToAsignaturaAlumno
- GET /alumnos/asignaturas getAlumnosYAsignaturas
- GET /asignaturas getAsignaturas
- POST /asignaturas addAsignatura
- GET /asignaturas/{acronimo} getAsignaturaPorAcronimo

At the bottom of the list, there's a red bar.

Eligiendo alguna de las operaciones se despliegan detalles con más información (en la ilustración se trata de GET sobre el URI /alumnos) y con la posibilidad de invocación (botón Try it out)

The screenshot displays the Swagger UI for the 'centro-educativo-http-controller' API. The selected endpoint is 'GET /alumnos getAlumnos'. The 'Parameters' section is empty, and the 'Responses' section shows a table of HTTP status codes and their descriptions. The 'Try it out' button is visible in the top right of the endpoint details.

Code	Description
200	OK
401	Unauthorized
403	Forbidden
404	Not Found

Below the table, an 'Example Value' is provided for the 200 response, showing a JSON array of student objects:

```
[
  {
    "apellidos": "string",
    "dni": "string",
    "nombre": "string",
    "password": "string"
  }
]
```

Other endpoints visible in the UI include:

- POST /alumnos addAlumno
- GET /alumnos/{dni} getAlumnoPorDNI
- DELETE /alumnos/{dni} delAlumnoPorDNI
- GET /alumnos/{dni}/asignaturas getAsignaturasDeAlumno

En la pestaña Models consultamos el modelo de datos:



### 7.3 Invocación con detalles HTTP (curl)

**curl** (*client url*) es una pequeña aplicación presente en todas las distribuciones UNIX mediante la que se pueden emitir peticiones HTTP y recibir respuestas. Siendo ése su propósito, dispone de múltiples parámetros y opciones para configurar cada interacción HTTP, obteniendo detalles vitales que necesitaremos considerar en el código de nuestro desarrollo.

Resumidamente: `curl -X MÉTODO_HTTP URL`

- Puedes consultar su sintaxis y opciones en cualquier sistema UNIX con `man curl`

Tres opciones destacables:

- Para que no muestre información sobre la transmisión: `-s`
- Para que muestre detalles internos, especialmente cabeceras: `-v`
- Para enviar datos codificados en JSON: `--data '{"propiedad": "valor"}' -H "content-type: application/json"`



Por tanto `curl` se encuentra en un punto intermedio entre el cliente `RESTED` anterior y el código Java que hemos de crear.

Además es muy interesante considerar que se puede escribir un script en el shell que, invocando la orden `curl` con los argumentos necesarios, pueda ejecutar una secuencia de operaciones sobre la aplicación `CentroEducativo` (p.ej. para crear 10 alumnos nuevos).

A continuación ilustraremos cómo llevar a cabo los dos pasos descritos en "Trabajando con `CentroEducativo`", para lo que necesitamos destacar estas opciones de `curl`:

- Para que no muestre información sobre la transmisión: `-s`
- Para que muestre detalles internos, especialmente cabeceras: `-v`
- Para enviar datos codificados en JSON<sup>7</sup>: `--data '{"propiedad": "valor"}' -H "content-type: application/json"`
- Para emplear un fichero `cucu` tanto para consulta como escritura: `-c cucu -b cucu`

Paso 1: **identificación** (suponiendo que el DNI es 23456733H y la clave 123456)

```
curl -s --data '{"dni":"23456733H","password":"123456"}' \
-X POST -H "content-type: application/json" http://dew-millogin-2223.dsicv.upv.es:9090/CentroEducativo/login \
-c cucu -b cucu
```

En pantalla se mostrará la clave de sesión que deberemos reutilizar más tarde. A partir de aquí supondremos que dicha clave se encuentra en la variable de entorno `KEY`

Paso 2: **operaciones**, que deben usar el mismo fichero de cookies y la misma clave de sesión que en el primer paso.

P.ej. para consultar la lista de alumnos y asignaturas:

```
curl -s -X GET 'http://dew-millogin-2223.dsicv.upv.es:9090/CentroEducativo/alumnosyasignaturas?key=$KEY \
-H "accept: application/json" -c cucu -b cucu
```

P.ej. para la lista de alumnos y calificaciones de DEW:

```
curl -s -X GET 'http://dew-millogin-2223.dsicv.upv.es:9090/CentroEducativo/asignaturas/DEW/alumnos?key=$KEY \
-H "accept: application/json" -c cucu -b cucu
```

P.ej. para añadir un alumno (los detalles se pueden ver en la propia orden):

```
curl -s --data '{"apellidos": "Soy nuevo En el barrio", "dni": "33445566X", "nombre": "Bienvenido", "password": "Inkreibler"}' \
-X POST -H "content-type: application/json" 'http://dew-millogin-2223.dsicv.upv.es:9090/CentroEducativo/alumnos?key=$KEY \
-c cucu -b cucu
```

Como se puede observar, esta última operación genera un error, pues `CentroEducativo` (a diferencia de lo que se exige en nuestra aplicación) sí que gestiona un rol administrativo. Ese rol administrativo resulta necesario para dar de alta nuevos alumnos y profesores. No tiene excesivo sentido que un profesor pueda añadir nuevos profesores.

La adición de alumnos y profesores únicamente será necesaria fuera de la aplicación web a desarrollar. La cuenta del rol administrativo utiliza como **dni** el valor "11111111" (9 unos) y como **password** "654321".

<sup>7</sup> ¡mucho cuidado con las comillas!

## 7.4 Invocación por programa (en Java)

Las herramientas anteriores sirven para hacer pruebas, pero éstas son una ayuda al desarrollo de un programa<sup>8</sup> que invoque esos servicios REST. Aquí deberemos preocuparnos por dos aspectos: el protocolo HTTP y la representación JSON.

### 7.4.1 Cómo construir una petición HTTP y recibir la respuesta

Es necesario para poder establecer comunicación por HTTP con CentroEducativo.

Se pueden encontrar múltiples opciones para este requisito: algunas usan Java "básico", otras dependen de la versión del lenguaje (p.ej. la serie 9 de Java incorpora una buena API `HttpClient`, y en la serie 11<sup>9</sup> mejora incluso más), y un grupo numeroso de desarrolladores aprovecha bibliotecas que se han ido creando con estos mismos objetivos, destacando:

- **JAX-RS Client**, que viene incluido en estándares JEE posteriores a la cinco.
- **OkHttp** (<https://square.github.io/okhttp/>)
- **HttpComponents** (<https://hc.apache.org/>), versión 5, de la fundación Apache. Comienza por <https://hc.apache.org/httpcomponents-client-5.0.x/quickstart.html>

**Atención.** El uso de bibliotecas externas siempre requiere cautelas por quién las usa:

1. ¿Tengo todos los requisitos exigidos por la biblioteca? Por ejemplo, `OkHttp` necesita `kotlin`.
2. ¿Qué versión de la biblioteca seleccionada es adecuada para mi aplicación?, ¿tiene problemas de estabilidad o seguridad?

La conclusión no puede ser otra que la necesidad de documentarnos suficientemente para responder a estas preguntas. También debemos recordar que esas bibliotecas han de formar parte del despliegue automatizado (archivo WAR).

### 7.4.2 Cómo codificar y decodificar datos en JSON

Como en el caso anterior puede emplearse Java básico cuando la estructura de la información no es compleja, o recurrir a bibliotecas. Destaco:

- **Gson** (<https://github.com/google/gson>) de Google
- **Json-java** (<https://github.com/stleary/JSON-java>) de [www.json.org](http://www.json.org)

Pueden aplicarse las mismas cautelas y conclusión acerca de bibliotecas que se han citado anteriormente. En ambos casos (HTTP y JSON) hay bastantes opciones. Un criterio útil consiste en seleccionar algún ejemplo completo que nos convenza, y adaptarnos a las bibliotecas que necesite.

<sup>8</sup> ¡como nuestros servlets!

<sup>9</sup> <https://dzone.com/articles/java-11-standardized-http-client-api>

## 8 GESTIÓN DE LA IDENTIDAD (AUTENTICACIÓN)

El control de la identidad es un requisito que puede ser considerado en cada uno de los niveles de nuestro desarrollo:

1. En primer lugar, el **contenedor Tomcat**, y cualquier servidor web, distingue e identifica usuarios con una implementación propia que opera a nivel HTTP. El concepto de rol nos sirve para tratar grupos de usuarios de manera uniforme.
2. En segundo lugar, la **API para servlets** dispone de sus propias clases para expresar el concepto de usuario y la forma en que éste debe demostrar su identidad. Normalmente no coincide con el caso anterior, pero debe ser compatible.
3. Por último, el **nivel de datos** también dispone de alguna implementación para poder distinguir entre accesos permitidos y prohibidos. Tampoco coincidirá con los casos anteriores.

Es un problema nuestro, del nivel de lógica de datos, conseguir que todo esto pueda funcionar conjuntamente. A continuación, dentro de este apartado, se indicarán detalles relacionados con la autenticación en Tomcat (es un asunto de configuración). Posteriormente se describirá cómo conseguir mediante sesiones la adaptación entre usuarios web y usuarios del nivel de datos.

### 8.1 Configuración de Tomcat

Si se van a dar 2 roles diferenciados (alumno y profesor), ¿cómo se puede configurar la aplicación para averiguar y comprobar de qué usuario se trata y a qué rol pertenece? A continuación se describe una parte de la mecánica necesaria.

Si deseamos que el servidor reciba garantías respecto a la identidad del cliente, **lo usual** es que le obligue a identificarse mediante un **nombre** y **contraseña**.

- En ausencia de cifrado, el envío de esas credenciales es preocupante en términos de privacidad; por ello resultaría altamente conveniente mejorar este apartado, pero no le dedicaremos el esfuerzo necesario.

Concretando, queremos **limitar el uso de la aplicación web** a un usuario (**cliente**), con una contraseña (**miKlave**). Para el envío de login y clave recurriremos al método de autenticación básico. Para ello necesitamos realizar estos 2 pasos:

1. Añadimos en **tomcat-users.xml** (directorio tomcat/conf) una entrada para el rol **mirol** y el usuario **cliente**:

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="mirol"/>
  <user username="cliente" password="miKlave" roles="mirol"/>
</tomcat-users>
```

Es un archivo de configuración global que se encuentra en el directorio conf de Tomcat. ¡Cuidado con la presencia de comentarios en el archivo que pueden **desactivar** las órdenes!

2. Y al final del `web.xml` de la aplicación web a proteger, añadimos las líneas destacadas en **negrita**:

```
...
<security-constraint>
    ...
    <auth-constraint>
        <role-name>mirol</role-name>
    </auth-constraint>
</security-constraint>

<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>Protegido</realm-name>
</login-config>
</web-app>
```

Este archivo de configuración está asociado a la aplicación. Podemos modificarlo en el proyecto desde Eclipse (recomendado para incluir los cambios en posteriores despliegues) o en el servidor tras cada despliegue de la aplicación.

Tras reiniciar el contenedor, al acceder al mismo URL que se ha mencionado, deberemos identificarnos con el nombre y contraseña en la ventana que el navegador nos muestra.

- En nuestro caso tenemos 2 roles (`rolpro`, `rolalu`) para agrupar profesores y alumnos respectivamente. Verás que el uso de **tomcat-users.xml** para cada uno es un completo disparate (pero te sirve para comenzar).

Esta primera barrera, la seguridad declarativa, es demasiado burda porque no nos permite, p.ej., comprobar que un alumno no accede a datos de otro, o que un profesor no modifica las calificaciones de una asignatura que no está bajo su responsabilidad. Basarse en la URL de los servlets no permite conocer ni controlar con qué argumentos están siendo invocados<sup>10</sup>.

Por ello es muy importante dentro de la API 2.5 el uso de los métodos `getRemoteUser()`, `isUserInRole()` y similares sobre el objeto petición del servlet: seguridad programática.

## 8.2 Compatibilizando sesiones web y de datos

Debemos diferenciar entre la autenticación entendida por el nivel de datos y la definida por nuestro nivel de lógica de la aplicación. Pueden estar asociadas pero no son idénticas a efectos de significado y alcance; por ello separamos un espacio de identificación válido entre usuario web y nivel de lógica, y otro entre nivel de lógica y de datos. Será necesario alguna forma de transitar entre uno y otro, y por ello mostramos aquí un **pseudocódigo** de referencia.

Propósito: garantizar, en caso de usuario autorizado, que los servlets puedan encontrar los valores `dni`, `password` y `key` en la sesión actual, para poder interactuar con el nivel de datos.

Condiciones de funcionamiento:

- Posible sesión abierta, con atributos `key`, `dni` y `pass`

<sup>10</sup> Es cierto que con cierto rediseño se pueden mejorar algunos aspectos, pero no termina de resolverse.

- API de servlets que permite el acceso a los datos de identificación mediante `getRemoteUser` e `isUserInRole`

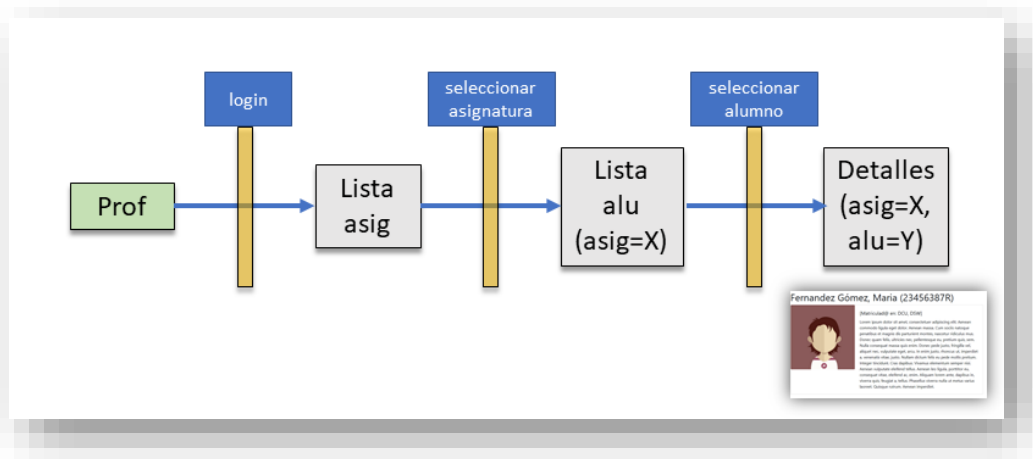
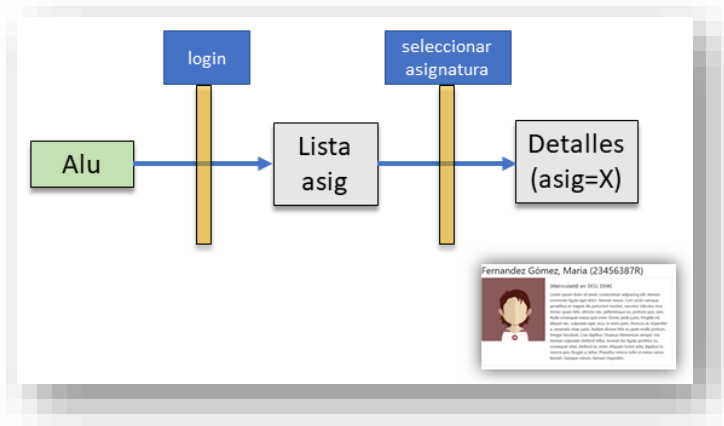
Detalles:

- `sesion` es la sesión actual, si la hay, permitiendo el acceso a los atributos antes mencionados.
- `web.auth()` y `data.auth()` modelan el resultado de la operación de autenticación en el nivel de lógica y en el de datos, respectivamente
  - El primero es implícito si la configuración de la aplicación (`web.xml`) exige dicha autenticación.
  - El segundo funciona invocando el URL `/login` de la aplicación `CentroEducativo`, pasando los parámetros adecuados y recibiendo el resultado en el cuerpo del mensaje HTTP de respuesta.
- La relación entre ambas autenticaciones se establece con un vector de usuarios (`users[]`), indexado por `login`. No almacenamos la clave de acceso puesto que forma parte de la sesión y es efímera.
- Las condiciones de error aparecen marcadas pero su tratamiento no ha sido detallado.
- Este fragmento con pseudocódigo puede ser implementado al comienzo de cada servlet o como un filtro

```
01: if (!sesion.key) {  
02:   if (login = web.auth()) { // usando getRemoteUser()  
03:     sesion.dni = users[login].dni;  
04:     sesion.pass = users[login].password;  
05:     if (key = data.auth(sesion.dni, sesion.pass)) { // invocando /login por POST  
06:       sesion.key = key;  
07:     }  
08:   } else { error("data.auth");  
09:   }  
10: } else { error("web.auth");  
11: }  
12: }  
13: // Continuar con invocación al servlet correspondiente
```

## 9 LA VISTA DEL USUARIO: NAVEGACIÓN, CONTENIDOS, ASPECTO

En los dos casos de uso apuntados en el apartado 7, salvo "operaciones derivadas", aparecen 3 grupos de páginas, tal y como se pueden observar en estas dos secuencias de navegación:



### 9.1 La página de bienvenida/login

Una página que anuncia el sitio web, y en la que aparece una opción para identificarse como alumno (enlace "consultar" tus calificaciones") o como profesor (enlace "consultar o modificar" calificaciones).

La ilustración es una referencia.

(Al ser un trabajo de asignatura, debe mostrarse información que en una web normal no aparecería)

## Bienvenid@ a Notas OnLine

Una aplicación que cuesta más de lo que parece para conseguir menos de lo que creías... ¿¡Qué más se puede pedir!?

**Si eres alumn@...**  
 Podrás [consultar](#) tus calificaciones... Debes contar con tus datos identificativos para acceder.

**Si eres profesor@...**  
 Podrás [consultar o modificar](#) las calificaciones en tus asignaturas... Debes contar con tus datos identificativos para acceder.

**Grupo g5\_lab0\_Miercoles\_1**

1. Apellidos y nombre
2. Apellidos y nombre
3. Apellidos y nombre
4. Apellidos y nombre
5. Apellidos y nombre
6. Apellidos y nombre

Trabajo en grupo realizado para la asignatura Desarrollo Web. Curso 2019-2020 (aka el curso del COVID-19, en toda la frente)

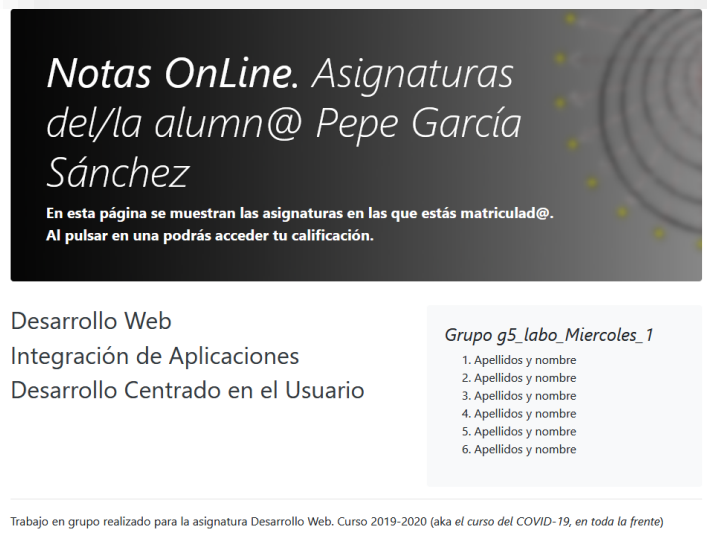
## 9.2 Las páginas de navegación intermedias

No tienen nada especial que destacar, salvo la personalización.

La ilustración de la derecha muestra una lista de las asignaturas (3 en este caso) en las que está matriculado un alumno (Pepe García Sánchez).

- Cada título de asignatura es un enlace para consultar la calificación del alumno.

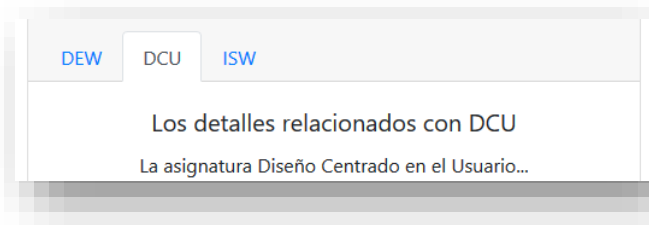
Las páginas que muestran la lista de asignaturas que imparte un profesor, o la de alumnos de una asignatura tienen un diseño similar.



### 9.2.1 Las páginas múltiples

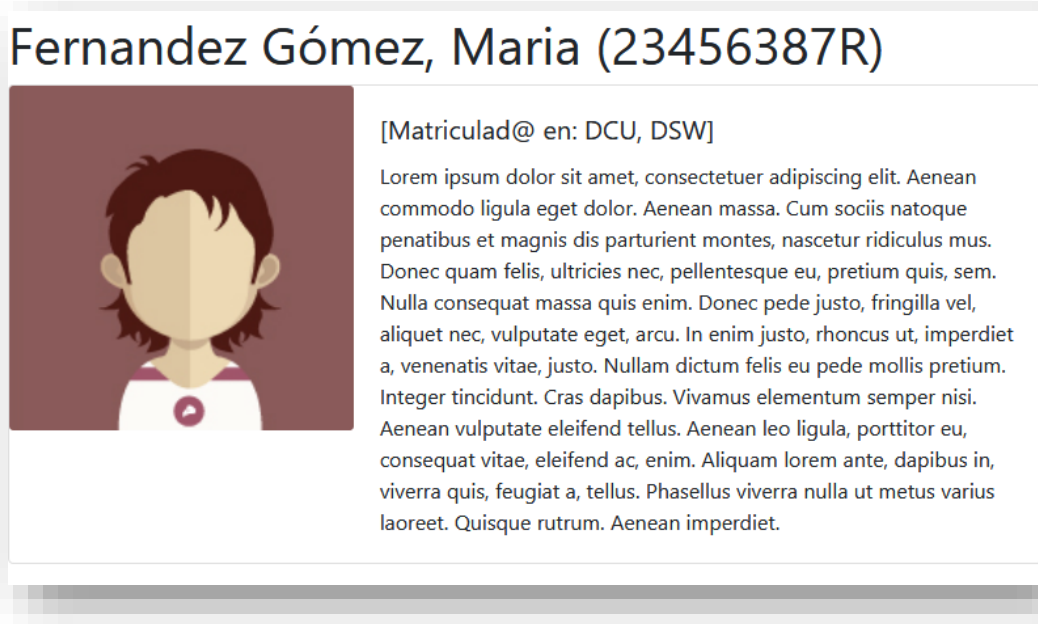
Permiten visualizar varios contenidos pasando de uno a otro sin abandonar aparentemente la página. Una implementación sencilla consiste en usar pestañas (*cards*) de Bootstrap (<https://getbootstrap.com/docs/4.6/components/card/>)

- Colocaríamos como pestaña cada asignatura en la que está matriculado un alumno, o que imparte un profesor.



### 9.3 Las páginas finales de detalle

Tienen un diseño algo más elaborado porque deben integrar informaciones más variadas.



El servlet construye la página final incluyendo **4 clases de información**:

1. El hipertexto necesario para construir una página basada en **Bootstrap**
2. Las informaciones recuperadas del **nivel de datos**, codificadas en JSON, tales como dni, nombre, apellidos, ...
3. Las que se obtienen **a partir de la información del nivel de datos**, p.ej. el dni se emplea como nombre del archivo que contiene la fotografía, que hay que devolver. También, en su caso, la puntuación media también forma parte de este tipo de información.
4. Informaciones **de relleno**, como el texto "Lorem ipsum". Es insertado automáticamente por el servlet.

La página de detalles mostrada contiene información del DNI, la lista de asignaturas en las que está matriculada la alumna, los apellidos y el nombre, el contenido de la imagen (contenido del archivo cuyo nombre coincide con el DNI) y el texto de relleno.

#### 9.3.1 El caso especial de la operación AJAX

En el apartado 7 ha sido mencionada la posibilidad de que el profesor pueda "*consultar o modificar la nota obtenida de cada uno de los alumnos en una de las asignaturas que imparte*". Se pretende agilizar la interacción cargando toda la información en el navegador (será una operación AJAX). A partir de ese momento todo el trabajo de consulta será local al navegador, y las modificaciones deberán llegar al servidor.

Esperamos que una **descripción de la parte JavaScript** dentro de esta página pueda aclarar el comportamiento.

0. Suponemos que la operación es invocada por un profesor de la asignatura asig.
1. Realizar una petición AJAX para recibir la lista de alumnos con sus calificaciones en asig.



- Se codificará en JSON, como un vector cuyo contenido se recorrerá para consultar cada elemento.

Puede optarse por...

- incluir un servlet específico para realizar toda la operación.
  - que la página solicite por AJAX uno a uno cada elemento, pero todos al principio (evento).
2. En este momento, dentro de la página, **debe haber una estructura de datos que contenga TODO el material necesario** para la navegación local.
    - ¡¡Incluyendo las fotografías!!
    - Es conveniente copiar el resultado *temporal* de la operación AJAX, posiblemente ligado a su clausura, a un vector mediante la orden `slice`.
  3. Visualizar los datos del/la primer/@ alumn@, con flecha a izquierda y derecha (eventos)
    - Las flechas permiten seleccionar el/la alumn@ a visualizar. Deben considerarse los casos especiales del primero y último.
  4. La **calificación debe ser una viñeta editable**. Si se coloca contenido (evento) en ella, deberá modificarse en el servidor.
    - Es indiferente que en el servidor se actualice siempre o únicamente al cambiar el valor.

## 10 PRIMERAS OPERACIONES SOBRE LA APLICACIÓN

Un ejemplo de desarrollo como éste cuenta con simplificaciones que alivian los posibles problemas que se puedan plantear. Un caso destacable es que la aplicación NO dispone de un soporte permanente para recordar los cambios realizados. La aplicación, al arrancar, reinicia todos sus datos y usa los contenidos preestablecidos que se muestran a continuación..

(Para todos los usuarios la clave es '123456')

### 10.1 Usuarios del rol pro

Los usuarios profesores que están dados de alta inicialmente en el nivel de datos son

```
[
  {
    "dni": "23456733H",
    "nombre": "Ramón",
    "apellidos": "Garcia"
  },
  {
    "dni": "10293756L",
    "nombre": "Pedro",
    "apellidos": "Valderas"
  },
]
```

```
{
  "dni": "06374291A",
  "nombre": "Manoli",
  "apellidos": "ALbert"
},
{
  "dni": "65748923M",
  "nombre": "Joan",
  "apellidos": "Fons"
}
]
```

### 10.2 Usuarios del rol alu

Los usuarios alumnos que están dados de alta inicialmente en el nivel de datos son:

```
[
  {
    "apellidos": "Garcia Sanchez",
    "password": "123456",
    "nombre": "Pepe",
    "asignaturas": [
      "DCU",
      "DEW",
      "IAP"
    ],
    "dni": "12345678W"
  },
  {
    "apellidos": "Fernandez Gómez",
    "password": "123456",
    "nombre": "Maria",
    "asignaturas": [
      "DCU",
      "DEW"
    ],
    "dni": "23456387R"
  },
  {
    "apellidos": "Hernandez Llopi",
    "password": "123456",

```

```
    "nombre": "Miguel",
    "asignaturas": [
      "DCU",
      "IAP"
    ],
    "dni": "34567891F"
  },
  {
    "apellidos": "Benitez Torres",
    "password": "123456",
    "nombre": "Laura",
    "asignaturas": [
      "IAP",
      "DEW"
    ],
    "dni": "93847525G"
  },
  {
    "apellidos": "Alonso Pérez",
    "password": "123456",
    "nombre": "Minerva",
    "asignaturas": [],
    "dni": "37264096W"
  }
]
```

## 10.3 Asignaturas

Las asignaturas creadas originalmente son:

```
[
  {
    "acronimo": "DEW",
    "nombre": "Desarrollo web",
    "curso": 3,
    "cuatrimestre": "B",
    "creditos": 4.5
  },
  {
    "acronimo": "IAP",
    "nombre": "Integración de Aplicaciones",
    "curso": 4,
    "cuatrimestre": "A",
    "creditos": 4.5
  },
  {
    "acronimo": "DCU",
    "nombre": "Desarrollo Centrado en el Usuario",
    "curso": 4,
    "cuatrimestre": "A",
    "creditos": 4.5
  }
]
```

Las primeras operaciones a construir únicamente realizan consultas al sistema, devolviendo en todos los casos una página de hipertexto.

## 10.4 Caso a resolver

El **primer caso a resolver** debe dar solución a este supuesto:

1. Una alumna se identifica con éxito.
  - Correcto: página que muestra la lista de *sus* asignaturas
  - Error: devolver código adecuado.
2. Seleccionando una asignatura, consulta su puntuación
  - Correcto: página que muestra la lista de alumnos

**Atención:** este primer caso incluye muchos y variados aspectos que deben ser objeto de estudio del grupo. Es una actividad muy abierta en la que debe mantenerse una mentalidad receptiva. Debes entender que posteriormente aparecerán muchos otros casos a resolver, algunos de los cuales pueden obligarte a replantearte o generalizar lo que ahora propongas.

Este caso forma parte de un hito, por lo que debe proporcionarse una documentación adecuada (cuando debas realizar una elección es imprescindible argumentar la razón), y capturas que actúen como muestras de buen funcionamiento. Usa un servidor de portal para validar su funcionamiento, e inclúyelo en el acta.

## 11 SELECCIÓN DE OPERACIONES

Las operaciones se organizan con bastante naturalidad en dos grandes grupos:

### 1) Operaciones del **ciclo de vida** (dar de alta o eliminar usuarios y asignaturas)

Estas operaciones del ciclo de vida están soportadas por el nivel de datos, pero **no las facilitaremos a través de la web**. En otras palabras, en la inicialización de nuestra aplicación, poblaremos el sistema<sup>11</sup> con los datos necesarios (alumnos, profesores, asignaturas), pero no ofreceremos esa funcionalidad a través de la web.

### 2) Resto de operaciones: se organizan dependiendo del número de posibles resultados, como

...

#### a) consultar detalles|modificar un elemento individual a partir de su id

- P.ej. consultar los datos personales de un alumno, dado su id
- P.ej. modificar la calificación de un alumno en una asignatura, dados sus ids

#### b) consultar todos los ids de un cierto colectivo

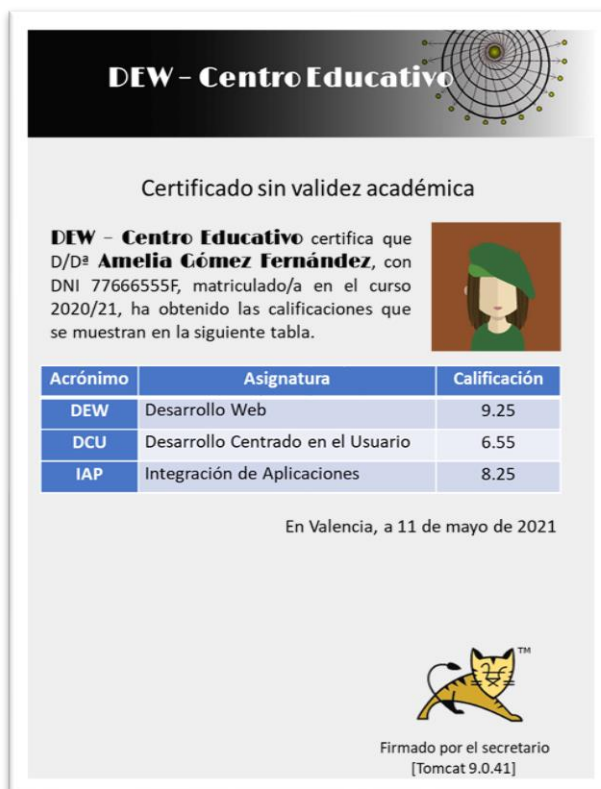
- P.ej. consultar los ids de los alumnos matriculados de una asignatura, dado el id de la asignatura

Combinando usuarios, operaciones y objetos aparecen cerca de 25 posibles operaciones, a las que luego pueden añadirse otras derivadas. Es un problema de combinatoria.

Sin embargo nosotros deseamos seleccionar únicamente las necesarias para que...

Una **alumna** pueda...

- Identificarse como una alumna concreta (inicio de sesión)
- Consultar la lista de asignaturas en las que está matriculada
- Consultar la nota obtenida en una de las asignaturas en la que está matriculada
- **Operación derivada:** crear una página formateada para su impresión, con una relación de asignaturas y calificaciones obtenidas, a modo de certificado, incluyendo en la página la fotografía de la alumna
- Finalizar la sesión



<sup>11</sup> En uno de los apéndices se muestra una forma automatizada de invocar las operaciones que permiten, entre otras cosas, poblar el sistema con usuarios

Un **profesor** pueda...

- Identificarse como un profesor concreto (inicio de sesión)
- Consultar la lista de asignaturas<sup>12</sup> que imparte. Se necesitará recorrer todas las asignaturas y anotar aquéllas en las que este profesor aparezca.
- Consultar la lista de alumnos en una de las asignaturas que imparte
- Consultar o modificar la nota obtenida por uno de los alumnos en una de las asignaturas que imparte
- **Operación derivada:** calcular la nota media de una de las asignaturas que imparte
- **Operación derivada AJAX:** interacción ágil para consultar o modificar la nota obtenida de cada uno de los alumnos en una de las asignaturas que imparte.
  - Debe cargar todos los datos en el navegador, facilitando el recorrido entre los alumnos y volcando las modificaciones (o todas las calificaciones) al servidor
  - Debe incluir la fotografía del alumno visualizado
- Finalizar la sesión

Estas características deben ser compatibles con limitaciones de seguridad que se resumen en...

Si algo no está permitido explícitamente, entonces está prohibido

Sin ser exhaustivo, pero buscando los detalles...

- Una alumna no puede *leer* datos de...
  - Otras alumnas
  - Ningún profesor
  - Asignaturas en las que no está matriculada
- Un profesor no puede *leer* datos de...
  - Otros profesores
  - Asignaturas que no imparte
  - Alumnas y alumnos no pertenecientes a asignaturas que imparte

Cambiando *leer* por *modificar*, todas las operaciones están prohibidas salvo

- Un profesor solo puede *modificar*...
  - Calificaciones en asignaturas que imparte
  - Calificaciones de alumnas en asignaturas que imparte
- Una alumna no puede modificar nada

Cualquier *otra operación*, si la hubiera, está prohibida. Esto incluye casos que nos puedan parecer innecesarios, como "Un usuario no identificado no puede hacer NADA". Estudia la siguiente comprobación:

---

<sup>12</sup> No se menciona como operación *derivada* porque no es una operación extra sino básica, pero no ofrecida por el actual modelo del nivel de datos

1. Si copio una URL desde un usuario, ¿puedo usarla desde otro? P.ej. la alumna María Fernández ¿puede usar la URL que le proporciona acceso a la nota de Miguel Hernández en DCU? (`/nol2223/Alumno?dni=34567891F&signatura=DCU`). Forma general de obtener estas URLs: conectar como Miguel Hernández, anotar las URLs que emplea, conectar ahora como María Fernández y observar qué ocurre si se usan esas mismas URLs.
2. Debe observarse que la ausencia de un enlace en una página no impide que un usuario pueda escribirla en la barra de direcciones y acceder a su resultado.

Otro aspecto que aquí tiene importantes consecuencias es el relativo a las diferencias sustanciales entre la implementación del concepto de usuario de la web, y la implementación del concepto de usuario del nivel de datos.

- La identificación necesaria en el nivel de datos debe ser construida y mantenida por el nivel de lógica, sin exigir al usuario ninguna otra identificación tras la inicial.

También en el nivel de lógica añadiremos alguna información extra no disponible en el nivel de datos: una **fotografía** de cada alumno. Se establecerá el criterio para seleccionar la fotografía correcta a partir del DNI de cada usuario.

- En el apartado 13 encontrarás información imprescindible sobre todo lo relacionado con las fotografías.

**Cada operación se implementa básicamente mediante un servlet**, dándonos la oportunidad de controlar su configuración y otras propiedades. No necesitamos ofrecer una interfaz REST, pero las operaciones en las que interviene alguna interacción ágil se implementan con una parte JavaScript, que requiere resultados JSON, en el cliente.

Un par de preguntas **con sustancia** que deben ser contestadas primero "en abstracto" y luego comprobadas en vuestra implementación; quizás no lo hayas considerado explícitamente pero puede haber varios clientes conectados simultáneamente a esta web. En esa situación...

- ¿En qué casos puede haber un conflicto de acceso tipo lectores/escritores?

Además puede darse la circunstancia de que dos usuarios observen valores diferentes para la misma información.

- ¿Puedes dar un ejemplo en el que esto pueda suceder?, ¿puede solucionarse?, ¿vale la pena?

## 12 AJAX: MÁS PIEZAS A DESARROLLAR

Tras el tema sobre servlets aparece otro dedicado a "Interacciones ágiles cliente/servidor" y un fragmento (la parte sobre AJAX) del seminario sobre jQuery. Esos conocimientos te permitirán desarrollar las piezas mencionadas en este apartado.

La web **TheCatAPI** ofrece un servicio (<https://api.thecatapi.com/v1/images/search>) que devuelve objetos JSON. Tomando un ejemplo de resultado, observamos que ...

- Su tipo MIME es `application/json`
- Su contenido (en un ejemplo cualquiera) es:

```
[
  {
    "breeds": [],
    "id": "bet",
    "url": "https://cdn2.thecatapi.com/images/bet.jpg",
    "width": 500,
    "height": 337
  }
]
```

Observa que se trata de un vector con un único elemento. Construimos una página (`gatoazar.html`) que inserte la imagen del atributo `url` en un `div`.

```
01: <!DOCTYPE html>
02: <html lang="es-es">
03: <head>
04: <title>gatoazar</title>
05: <script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
06: <script>
07: $(document).ready(function() {
08:   $.ajax({
09:     url: 'https://api.thecatapi.com/v1/images/search',
10:     dataType: 'json',
11:     success: (data) => {
12:       // solo un gato, pero dentro de un vector
13:       let estegato=data[0]
14:       $('<div class="lista-gatos"></div>').append('')
15:     })
16:   })
17: })
18: </script>
19: <!-- Tambien puedes definir estilos para fotogato y lista-gatos -->
20: </head>
21: <body>
22:   <h1>Un gato</h1>
23:   <h3>Vaya minino! <small>Procedente de <code>thecatapi</code>!</small></h3>
24:   <div class="lista-gatos"></div>
25: </body>
26: </html>
```

### Un gato

/aya minino! Procedente de thecatapi!



### 13 ORGANIZACIÓN Y TRANSFERENCIA DE LAS IMÁGENES

Es bastante común que los SGBD relacionales no sean usados para representar y almacenar material no indexable de cierto volumen, como pueden ser vídeos o imágenes. En su lugar se puede emplear el sistema de ficheros, estableciendo...

- la ruta de un directorio en el que se encontrarán las imágenes
- el criterio por el que un nombre de fichero corresponde a una imagen concreta

Es posible que, por comodidad, el directorio seleccionado se encuentre dentro del espacio accesible por HTTP, de manera que exista un URL asociado al fichero que contiene la imagen; pero esta mecánica dificulta la limitación de acceso: ¿cómo determinar a qué imágenes puede acceder el profesor de una asignatura? Veréis que no se trata de un problema trivial en absoluto.

- Aquellos casos, incluyendo **TheCatAPI**, en los que la información devuelta incluye un URL de una imagen, entran en este grupo.

Como **alternativa**, si hemos determinado qué operaciones (servlets) son invocables por cada usuario, podríamos colocar en dichos servlets la capacidad para acceder a dichos archivos (que escaparían del acceso por URL) y devolver dinámicamente su contenido. En resumen: las imágenes NO se encuentran en un directorio accesible por HTTP.

Si la capacidad para consultar los datos de un alumno incluye el acceso a su fotografía, entonces podemos establecer como criterio que la fotografía de un alumno con DNI ABCD se encuentra en el fichero ABCD.png. Al fin y al cabo el DNI es único, ¿no?.

En la zona de recursos dispones de un archivo `fotos.zip` con los siguientes contenidos:

- Hay dos carpetas: `h` (hombres) y `m` (mujeres) de las que seleccionáis las fotos (realmente son avatares) a usar en función del género. Cada archivo seleccionado se renombra con el DNI de un usuario y se coloca en el lugar adecuado.
- Hay dos formatos, aunque realmente se debería usar solo el segundo:
  - Archivos `png`. Pueden ser empleados unitariamente pero no pueden transmitirse con facilidad como parte de otro contenido. Están diseñados para acceder directamente por HTTP. Solamente se usa esta alternativa si NO podéis resolver el problema con la otra opción, pero tiene una **reducción**
  - Archivos `pngb64`. Pueden ser transmitidos si en el destino se sabe cómo procesarlos (están codificados en base64). Ésta es la alternativa oficial a usar.

A continuación os mostramos un ejemplo del código del método GET de un servlet que devuelve un objeto JSON que contiene el DNI y la imagen recuperada del fichero (formato PNG, codificado en base64 para que su contenido pueda manejarse como una cadena sin "símbolos raros")

```
01: protected void doGet(HttpServletRequest req, HttpServletResponse res) throws
    ServletException, IOException {
02:
03:     String f = req.getParameter("dni");
04:     String carpeta = getServletContext().getInitParameter("directorio_imagenes");
05:     res.setContentType("text/plain");
06:     res.setCharacterEncoding("UTF-8");
07:     BufferedReader origen = new BufferedReader(new FileReader(carpetas+"/"+f));
08:     res.setContentType("text/plain");
```



```

09: PrintWriter out = response.getWriter();
10: out.print("{\"dni\": \""+f+"\", \"img\": \"\"}"); // Hay complicaciones con las comillas
11: String linea = origen.readLine(); out.print(linea); // Y con los saltos de línea!!
12: while ((linea = origen.readLine()) != null) {out.print("\n"+linea);}
13: out.print("\n"}");
14: out.close(); origen.close();
15: }

```

La invocación del servlet y su resultado son compatibles con el siguiente fragmento<sup>13</sup> AJAX, en el que se consulta un DNI concreto obteniendo un resultado cuyo atributo dni se usa como contenido de un elemento con id="esto", y la imagen codificada en el tributo img se coloca (tras un prefijo) como valor del atributo src (de una imagen con id="aqui")

```

$.getJSON("elServlet?dni=12345678w")
.done(function(response){
    $("#esto").text(response.dni);
    $("#aqui").attr("src", "data:image/png;base64,"+response.img);
})
.fail(function(jqxhr, textStatus, error) {
    var err = jqxhr.response.replace(", ", "\n"); // Pequeños ajustes
    alert("Algo mal: "+error);
});

```

## 14 EVALUACIÓN

La calificación de esta actividad se compone de 3 aspectos MUY diferenciados.

### 14.1 Hitos y entrega final

A lo largo del desarrollo de la actividad se marcan objetivos e instantes de tiempo en los que éstos deben haber sido cubiertos. Esos hitos han sido marcados en el documento “**DEW-Organización del Laboratorio 2**” publicado a finales de marzo. El propio proyecto final tiene también una fecha de entrega. En resumen:

	Fecha grupos Lunes	Fecha grupos Miércoles	Porcentaje nota
<b>Hito 1</b>	15/05	10/05	20%
<b>Hito 2</b>	29/05	24/05	30%
<b>Entrega final</b>	12/06	12/06	50%

Los retrasos sobre las fechas se penalizan:

- Retraso hasta una semana: -20%
- Retraso hasta dos semanas: -50%

Todos los hitos y la entrega final incluyen las actas hasta la fecha.

#### 14.1.1 Entrega final

Toda la documentación, incluyendo la memoria explicativa y las actas mencionadas anteriormente, estará en formato **Markdown**. Hay múltiples editores con visualización del aspecto resultante, como la extensión “**Instant Markdown**” (dbankier.vscode-instant-markdown) para Visual Studio Code. Como alternativa, también existen servicios gratuitos con

<sup>13</sup> Se emplean los métodos `done` y `fail`, que manejan la función `getJSON` como promesa.

editores online (<https://dillinger.io> es uno de ellos). También hay muchos lugares con documentación, como <https://www.markdownguide.org/getting-started/> (destacando <https://www.markdownguide.org/basic-syntax/>) y <https://about.gitlab.com/handbook/markdown-guide/>.

La elección de este formato nos ofrece la oportunidad de practicar con un *pariente* de HTML muy empleado en ciertos contextos, destacando la facilidad para insertar fragmentos de código HTML y JavaScript en lugar de capturarlos como imagen.

El entregable está constituido por dos piezas:

- 1) Un **documento** explicativo del trabajo realizado, que informe explícitamente sobre el nivel de consecución de cada elemento o versión. En este documento se debe...
  - Incorporar la identificación del número de grupo y turno de laboratorio, como, p.ej. 3TI12\_g1
  - Especificar la máquina de portal en la que se ha dejado instalada la aplicación (tipo dew-milogin-2223.dsicv.upv.es). Debe aportarse la información necesaria para poder acceder a dicho servidor para arrancar o detener la/s aplicación/es
  - Incluir comentarios de las partes significativas del código y/o configuración, siendo preferible mucho menos detalle en las partes menos significativas.
  - Incluir un apartado para explicar cómo interactúa cada código JavaScript con los servlets por AJAX, y otro para explicar cómo se insertan las informaciones en las páginas.
  - Explicar cómo se resuelve el apartado sobre las anotaciones de accesos (logs), mostrando la parte relevante para los accesos del apartado 7.
  - Incorporar, como anexos, las actas producidas en el desarrollo de esta actividad
- 2) Una *aplicación web* (no l2223.war), con todas las dependencias cubiertas, incluyendo el código fuente, instalada en tu servidor virtual **dew-milogin-2223.dsicv.upv.es**.
  - Es **imprescindible** que la aplicación web (no l2223.war) incluya la configuración necesaria y se encuentre desplegada sobre el servidor virtual con las restricciones especificadas.
  - El servidor no debe dejarse en funcionamiento.

Forma de entrega y plazo: se programará una tarea (DEW 2223-Trabajo NOL) para este propósito, con **cierre el 12 de junio a última hora**.

La calificación de los hitos y el entregable es una nota entre 0 y 10 a la que denominamos **HE**, y que actúa como valor a ponderar por los elementos que se mencionan a continuación.

## 14.2 Prueba individual (selectiva) corresponsable

Se trata de una prueba de laboratorio que evalúa a 2 de los miembros de cada grupo como representación de todo el equipo: uno es propuesto por cada equipo, y otro se selecciona al azar **en el mismo momento de la prueba**. La calificación obtenida (**PS**:  $0.7 \leq PS \leq 1.2$ ) puede decrementar o incrementar la calificación **HE**, en un intervalo [-30%, +20%]. Los alumnos examinados obtienen su nota, pero sus compañeros el promedio.

### 14.3 Coevaluación

En el momento del segundo parcial de la asignatura cada alumno repartirá 100 puntos entre sus compañeros, en reconocimiento del esfuerzo y las aportaciones que hayan realizado. La calificación obtenida (**CO**:  $0.7 \leq CO \leq 1.2$ ) puede decrementar o incrementar la calificación **HE**, en un intervalo  $[-30\%, +20\%]$ .

La nota correspondiente a este laboratorio 2 será una combinación en la que, partiendo de la calificación **HE** que procede de los hitos y la entrega final, cada alumno verá personalizada su nota en función de la prueba individual y de la coevaluación con un máximo de 10:

$$\text{Nota Labo 2} = \min(10, HE * PS * CO)$$

## 15 BIBLIOGRAFÍA

*[A lo largo del texto han aparecido múltiples enlaces que no se repiten aquí]*

"How to Call RESTful API Web Service in Servlet", 2016,  
<https://www.codeproject.com/Articles/1157120/How-to-Call-RESTful-API-Web-Service-in-Servlet>

- Menciona cómo "desentrañar" los detalles de una API REST externa, empleando además la herramienta jsonschema2pojo (<http://www.jsonschema2pojo.org/>), integrando en una aplicación sobre Eclipse, e incorporando la biblioteca Gson (<https://search.maven.org/artifact/com.google.code.gson/gson/2.8.6/jar>).

Para l@s muy valientes, aquí tenéis la oportunidad de interactuar con APIs externas:  
<https://medium.com/@anoopm6/a-list-of-rest-web-services-to-try-out-for-free-63a641ba7dab>

## 16 ANEXO: SERVLETS ACTUANDO COMO CLIENTES HTTP. OPERATIVA BÁSICA Y GESTIÓN DE COOKIES

Cuando nuestro código Java debe consumir servicios REST, como el facilitado por CentroEducativo, ha de preocuparse de la gestión de la conexión y de la codificación de la información intercambiada, principalmente representada mediante JSON. A lo largo del documento se citan algunas bibliotecas que pueden ser de ayuda. En este ejemplo gestionaremos la conexión mediante `HttpURLConnection`, y para la codificación emplearemos `Gson`<sup>14</sup>. Dado que ya se ha mencionado la web **TheCatAPI**, crearemos un servlet (MyCat) que aproveche dicho servicio. Pese a su sencillez, ilustra los pasos necesarios para...

- 1) Establecer la conexión
- 2) Enviar la petición
- 3) Procesar la respuesta (objeto JSON)
- 4) Crear una página de hipertexto con el URL de la foto

```
01: import java.io.*;
02: import java.net.HttpURLConnection;
03: import java.net.URL;
04: import java.util.Properties;
05:
06: import javax.servlet.ServletException;
07: import javax.servlet.http.HttpServlet;
08: import javax.servlet.http.HttpServletRequest;
09: import javax.servlet.http.HttpServletResponse;
10: import javax.servlet.http.HttpSession;
11:
12: import com.google.gson.Gson;
13:
14: /**
15:  * Servlet implementation class MyCat
16:  */
17: public class MyCat extends HttpServlet {
18:
19:     public class CatData { // Estructura de las entradas en el objeto JSON de respuesta
20:         public Object[] breeds;
21:         public Category[] categories;
22:         public String id;
23:         public String url;
24:         public int width;
25:         public int height;
26:     }
27:
28:     public class Category {
29:         public int id;
30:         public String name;
31:     }
32:
33:     private static final long serialVersionUID = 1L;
34:     private static final Gson gson = new Gson();
35:     private static final String catURL = "https://api.thecatapi.com/v1/images/search";
36:     private static final String preface = "<!DOCTYPE html>\n<html lang='es'>\n" +
37:         "<head>\n<meta charset='utf-8' />\n" +
38:         "<meta http-equiv='Content-Type' content='text/html' />\n" +
39:         "<title>Mi Gato</title>\n</head>\n<body>\n" +
40:         "<h1>Foto de un gato</h1>\n<h2>Recarga para ver otros...</h2>";
```

<sup>14</sup> Es importante exportar la biblioteca Gson como parte del proyecto (copiar el archivo jar a WEB-INF/lib)

```

41:  /**
42:   * solo implementamos doGet
43:   */
44:   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
45:       CatData[] cat;
46:       PrintWriter out = response.getWriter();
47:       response.setContentType("text/html");
48:       try {
49:
50:           URL urlas = new URL(catURL);
51:           HttpURLConnection con2 = (HttpURLConnection) urlas.openConnection();
52:
53:           con2.setDoOutput(true);
54:           con2.setRequestMethod("GET"); // Sin variables de formulario ni otras informaciones
extra
55:           con2.setRequestProperty("accept", "*/*");
56:
57:           try(BufferedReader br = new BufferedReader(
58:               new InputStreamReader(con2.getInputStream(), "utf-8"))) {
59:               String response3 = "";
60:               String responseLine = null;
61:               while ((responseLine = br.readLine()) != null) {
62:                   response3 += responseLine.trim();
63:               }
64:               cat = gson.fromJson(response3, CatData[].class);
65:           }
66:       } catch (Exception e) {
67:           response.sendError(500, "Hubo problemas al recuperar la información." + e);
68:           return;
69:       }
70:       if (cat != null) {
71:           out.println(preface);
72:           out.println("<div><img src=\"\" + cat[0].url + \"\" alt=\"el gato\" /></div>");
73:           out.println("</body></html>");
74:       }
75:   }
76: }

```

Otro aspecto de importancia para nuestras interacciones con CentroEducativo desde el código de los servlets es el mantenimiento de las sesiones del nivel de datos mediante **cookies**. De nuevo debemos programar en Java algo que un navegador gestiona automáticamente. Empleando el soporte nativo, mostramos código para dos momentos:

- Obtención de las cookies en la primera conexión

```

URLConnection connection = new URL(url).openConnection();
List<String> cookies = connection.getHeaderFields().get("Set-Cookie");

```

- Reenvío de cookies en cada conexión posterior

```

connection = new URL(url).openConnection();
for (String cookie: cookies) {
    connection.addRequestProperty("Cookie", cookie.split(";", 2)[0]);
}

```