Sistemas Orientados a Servicios - 2019

Práctica: Diseño e implementación de un servicio web RESTful

Es común encontrar una dificultad (cuando no reticencia) inicial a la hora de abordar un diseño RESTful de un servicio web. El principal problema recae en que REST no es algo tangible como pueda ser una especificación o una API. En su lugar, REST denota un estilo arquitectónico, esto es, un conjunto nombrado de restricciones sobre la interacción de diferentes componentes de la arquitectura de la Web que, cuando se respeta, dota a la arquitectura resultante de determinadas propiedades arquitectónicas deseables (por ejemplo, maximiza el incremento de la información identificada dentro del sistema). REST puede verse, por tanto, desde un punto de vista práctico, como un conjunto de mejores prácticas de diseño arquitectónico construidas sobre los estándares de http, URI y los principios arquitectónicos de la Web.

Como consecuencia, muchos servicios web y muchas API que se denominan RESTful no lo son en realidad, ya que sus diseños no siguen esas mejores prácticas y caen en errores como los siguientes:

a) Los objetos relevantes no se exponen como recursos, por lo que no podemos acceder a ellos directamente.
b) Los métodos HTTP no se utilizan correctamente. En su lugar todo se realiza mediante GET (incluso las operaciones que producen cambios) o POST.
c) Las representaciones de los recursos no están interconectadas, por lo que no puede "navegarse" desde un recurso dado al resto de recursos que conceptualmente deberían estar ligados al primero.

Estas APIs no pueden denominarse RESTful. En su lugar, deberían ser tratadas como meras APIs POX-RPC en las que se envían y reciben mensajes XML "planos" (sin ensobrado SOAP) mediante peticiones al estilo RPC sobre un protocolo de "transporte" http, utilizando un conjunto de parámetros opcionales en la cadena de consulta que influyen en los resultados obtenidos. Se trata de APIs equivalentes a las ofrecidas por los servicios SOAP o XML-RPC[1].

Teniendo en mente todo lo anteriormente comentado, esta práctica se plantea atendiendo a dos objetivos fundamentales:

a) Que el alumno aborde el diseño RESTful de un servicio sencillo, aplicando las mejores prácticas y las recomendaciones explicadas en las sesiones presenciales de la asignatura.
b) Que el alumno practique con el entorno Eclipse + Tomcat y la implementación de referencia de la API JAX-RS (Jersey) que soporta ese entorno construyendo una implementación prototipo del servicio diseñado en el paso anterior. Por

[1] Como caso de ejemplo de lo comentado anteriormente, se recomienda al alumno que revise las críticas a diseños de APIs como la de del.icio.us tratadas en clase. De igual forma, para el diseño del servicio propuesto se recomienda al alumno que revise las mejores prácticas y las recomendaciones comentadas en clase, así como el ejemplo de diseño propuesto para la API RESTful de un servicio similar a del.icio.us.

último, se plantea al alumno el desarrollo de un cliente sencillo que sirva de prueba del servicio implementado.

Ejercicio propuesto

Se trata de diseñar e implementar una API REST y un prototipo funcional de un servicio sencillo para una red social de tipo Facebook, donde los usuarios puedan interactuar entre ellos de diversas formas. Se asume que la autenticación y seguridad de la herramienta está realizada y por tanto no hay que implementarla.

En este servicio los usuarios publican mensajes en su página personal y pueden ser amigos de otros usuarios (relación de amistad recíproca) para poder ver los mensajes de éstos, a los que además pueden enviar mensajes privados (que no puede ver nadie más que los implicados en el envío, emisor y receptor). El servicio debe soportar a través de esa API las siguientes operaciones:

- Añadir un nuevo usuario a la red.
- Ver los datos básicos de un usuario.
- Cambiar datos básicos de nuestro perfil de usuario (excepto nombre de usuario).[2]
- Obtener una lista de todos los usuarios existentes en la red social[3]. Esta lista debe permitir ser filtrada por patrón de nombre (eg. Buscar todos los usuarios que contengan "Mar" en su nombre, "Mario", "María"…etc.)
- Publicar un nuevo mensaje en la página personal de un usuario.
- Eliminar un mensaje propio.
- Editar un mensaje propio.
- Obtener una lista de todos los mensajes de un usuario en su página personal. Además, esta lista debe permitir la opción de ser filtrada por fecha o limitar la cantidad de información obtenida por número de mensajes (e.g. los 10 primeros elementos, los elementos entre el 11 y el 20, etc.)
- Añadir un nuevo amigo
- Eliminar un amigo
- Obtener una lista de todos nuestros amigos. Además, esta lista debe permitir la opción de ser filtrada por el patrón de nombre o limitar la cantidad de información obtenida por número de amigos (e.g. los 10 primeros elementos, los elementos entre el 11 y el 20, etc.)
- Enviar un mensaje personal a otro usuario.
- Borrar nuestro perfil de la red social.
- Obtener una lista con los últimos mensajes de las páginas de nuestros amigos ordenados por fecha (similar a como lo muestra Facebook). Esta lista debe permitir la opción de ser filtrada por la búsqueda de contenido de texto (patrón) en el mensaje.

[2] Ciertas operaciones sobre mensajes, consultas de amigos, etc., obviamente sólo pueden realizarse sobre mensajes propios, amigos propios, etc. Sin embargo, la práctica no pide que se implemente ningún tipo de autenticación ni de autorización. Basta con que se identifique al usuario que está realizando la operación (i.e. al que se refiere la operación) a través de un "path param", un "query param" o del cuerpo del mensaje.

[3] Tanto esta lista como todas las demás, deben cumplir con los criterios de paginación navegabilidad vistos en clase.

- Consultar fácilmente la descripción necesaria para una aplicación móvil que queremos realizar, que muestre los datos básicos de un usuario, su último mensaje en la página, el número de amigos y los 10 últimos mensajes de las páginas de sus amigos que se han actualizado.

Implementar un cliente Java que pruebe ese servicio y por tanto, pruebe todas las operaciones anteriormente descritas (haciendo uso de los filtros en los casos necesarios).

Para la realización de la práctica se recomienda la siguiente organización del trabajo:

Paso 1. Diseño del servicio RESTful paso a paso:

a) Identifique todos los recursos en un modelo de recursos para el servicio, incluyendo recursos de información, colecciones, recursos compuestos, contenedores/fábricas, controladores, etc.
b) Diseñe los identificadores URI y patrones de identificación de todos los recursos en el modelo, siguiendo las convenciones vistas en clase.
c) Diseñe, para cada recurso, el subconjunto del interfaz uniforme de HTTP que ofrece. Incluya para cada verbo soportado por un recurso una breve descripción de su uso (por ejemplo, cuando se permita PUT para que el cliente pueda decidir y asignar el URI de un nuevo recurso creado, indique cómo debe ser ese nuevo identificador).
d) Diseñe los nuevos tipos de documentos XML ó JSON necesarios para el servicio. Diseñe los XML Schema ó JSON schema para esos nuevos formatos de documento XML. No es necesario proporcionar los esquemas, pero sí ejemplos de datos de todos los tipos de documentos definidos.
e) Resuma el diseño del servicio utilizando la siguiente tabla **por cada recurso definido y método soportado** para dicho recurso.

URI	Patró de URI del recurso	
Método	GET / POST / PUT / DELETE	
Cadena de consulta (sólo GET)	param1 = paramN=	Descripción del parámetro y del conjunto de valores posibles Etc.
Cuerpo de la petición (sólo PUT ó POST)	POX (tipo MIME o application/XML + referencia al XMLSchema, etc.)	
Devuelve	200 401 Etc.	OK +POX (tipo MIME o application/XML + referencia al XMLSchema, etc.) ó JSON Unauhtorized Etc.

Alternativamente, se podrá documentar cada recurso/método utilizando un fichero de Swagger Editor (versión de escritorio, no SwaggerHub), en cuyo caso se ofrecerá el fichero YAML, y se añadirán capturas de la versión web de la api de la documentación de cada recurso/método en la documentación.

Paso 2. Implementación de la API REST del servicio utilizando la implementación de referencia del estándar de soporte para REST JAX-RS (The Java API for RESTful Web

Services, JSR311[4]).

Así mismo, se implementará un prototipo funcional básico del servicio que permita mostrar su funcionamiento básico. Dicho prototipo deberá contar con algún mecanismo de persistencia de datos, recomendando el uso de una sencilla bases de datos SQL que facilitaría la construcción de consultas. Indicar brevemente el método de persistencia de datos seleccionado, así como el diagrama de entidad-relación o similar utilziado.

Paso 3. Implementación de un cliente Java que pruebe ese servicio y que realice al menos las operaciones anteriormente referidas.

La entrega consistirá en cuatro archivos

1. Memoria de la práctica (PDF). La memoria incluirá al menos:
    1. Resumen del diseño del servicio, incluyendo las tablas anteriormente referidas[5] y el diseño de la persistencia de los datos del servicio (Ej: BBDD, sistema de ficheros, estructura de objetos en memoria, etc.)
    2. Capturas de la ejecución de las operaciones anteriormente referidas desde un cliente REST (tipo Postman ó http Rest Client) en las que **pueda verse tanto los detalles de la invocación de la operación como los detalles del resultado de esa invocación**.
    3. Capturas de la ejecución del cliente de prueba para las operaciones anteriormente referidas
2. Datos utilizados para la prueba del servicio: si se ha optado por utilizar una BBDD, incluir un fichero .txt con el código SQL de creación de la BBDD, con las sentencias INSERT que considere necesarias para la correcta ejecución de su cliente, usuario y password requerido por la BBDD y detalle del nombre y versión del gestor utilizado, si se han utilizado ficheros, incluir el/los fichero(s) en el/los que se almacenan los datos, etc.
3. WAR **con fuentes[6]** generado en Eclipse con el código del servicio (WAR)
4. Proyecto con el código del cliente (comprimido ZIP ó RAR)

FECHA ENTREGA: 12 abril 2019

[4] http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html)

[5] Si se desea utilizar Swagger Editor, adjuntar capturas de las descripciones de cada método/verbo en la memoria, y adjuntar el fichero YAML

6 Si no se adjuntan las fuentes del servicio (y del cliente) la práctica no será corregida