Sistemas Orientados a Servicios - 2023

Práctica: Diseño e implementación de un servicio web RESTful

REST define un estilo arquitectónico, un conjunto de restricciones sobre la interacción de diferentes componentes de la arquitectura de la Web. REST puede verse, 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 de la Web.

Como consecuencia, muchos servicios web y muchas APIs 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 se puede acceder a ellos directamente.
b) Los métodos HTTP no se utilizan correctamente. En su lugar, todas las operaciones se realizan 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. Teniendo en cuenta estos aspectos, 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 clases de la asignatura.

b) Que el alumno implemente ese servicio sencillo usando el entorno Eclipse + Tomcat y la implementación de referencia de la API JAX-RS (Jersey).

c) Que el alumno desarrolle un cliente sencillo para probar el servicio implementado.

Ejercicio propuesto

Se trata de diseñar e implementar una API REST y un prototipo funcional de un servicio sencillo que simule 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 para poder ver los mensajes de éstos, a los que además pueden enviar mensajes. El servicio debe soportar a través de esa API las siguientes operaciones:

- Añadir un usuario nuevo a la red.
- Ver los datos básicos de un usuario.
- Cambiar datos básicos de nuestro perfil de usuario.[1]
- Borrar nuestro perfil de la red social.
- Obtener una lista de todos los usuarios existentes en la red social. Esta lista debe permitir ser filtrada por patrón de nombre (eg. Buscar todos los usuarios que contengan "Mar" en su nombre, "Mario", "Maria"…etc.).[2]
- Un usuario puede publicar un nuevo mensaje en su página personal.
- Un usuario puede eliminar un mensaje de su página personal.
- Un usuario puede editar un mensaje de su página personal.
- Obtener una lista de todos los mensajes escritos por 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 (e.g. los 10 primeros elementos, los elementos entre el 11 y el 20, etc.)
- Añadir un nuevo amigo de entre los usuarios registrados en la red social.
- 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 al perfil de otro usuario dentro de la red social.
- Consultar los últimos mensajes publicados de nuestros amigos en su página personal: obtener una lista de los últimos mensajes de todos mis amigos, pudiendo filtrar estos mensajes por fecha (último antes de cierta fecha). Poder limitar la cantidad de información obtenida por número.
- Buscar en todos los mensajes de nuestros amigos por contenido –contiene un determinado texto-. Poder limitar la cantidad de información obtenida por número.
- Consultar fácilmente la descripción necesaria para una aplicación móvil que queremos realizar, que muestre los datos básicos de un usuario, su último mensaje de su página, el número de amigos y los mensajes de sus 10 últimos amigos que se han actualizado (escritos por ellos mismos en su página personal de la red social).

[1] Ciertas operaciones como cambiar datos personales de un usuario, añadir amigos, escribir mensajes a otro amigo…etc. solo pueden realizarlas el propio usuario (o incluso un usuario administrador). 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.

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

Se pide:

a) Diseño del servicio RESTful. Se recomienda seguir estos pasos:

- Identifique todos los recursos en un modelo de recursos para el servicio, incluyendo recursos de información, colecciones, recursos compuestos, contenedores/fábricas, controladores, etc. Diseñe los nuevos tipos de documentos XML o JSON necesarios para el servicio. Diseñe los esquemas XML o JSON. No es necesario proporcionar los esquemas, pero sí ejemplos de datos de todos los tipos de documentos definidos.
- Diseñe los identificadores URI (*endpoint*) de todos los recursos en el modelo, siguiendo las convenciones vistas en clase.
- Diseñe, para cada recurso, el subconjunto del interfaz uniforme de HTTP que ofrece. Incluya para cada verbo empleado por un recurso una breve descripción de su uso.
- Resuma el diseño del servicio utilizando la siguiente tabla **para cada recurso definido y método soportado** para dicho recurso.

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

b) 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[3], JSR339[4] o JSR370[5]).

El prototipo deberá contar con algún mecanismo de persistencia de datos, recomendando el uso de una base de datos SQL que facilitaría la construcción de consultas. Indicar brevemente el método de persistencia de datos seleccionado (base de datos relacional, no relacional, ficheros, memoria...etc), así como el diagrama de entidad-relación o similar utilizado.

c) Implementación de un cliente Java que pruebe ese servicio y que llame a todas las operaciones del servicio. No es necesario realizar un cliente con interfaz gráfico o web, bastaría con un cliente que pruebe todas las operaciones realizadas o que tenga un menú que no permita ir probándolo.

[3] https://jcp.org/en/jsr/detail?id=310
[4] https://jcp.org/en/jsr/detail?id=339
[5] https://jcp.org/en/jsr/detail?id=370

La entrega se realizará en Moodle y contendrá cuatro ficheros:

1. Memoria de la práctica (PDF). La memoria debe incluir al menos la siguiente información:

    1. Resumen del diseño del servicio, incluyendo las tablas con el resumen de las operaciones del servicio[6] y el diseño de la persistencia de los datos del mismo (Ej, base de datos relacional o no relacional, sistema de ficheros, estructura de objetos en memoria, etc.)

    2. Capturas de la ejecución de las operaciones 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 base de datos, incluir un fichero .txt con el código SQL de creación de la base de datos, con las instrucciones INSERT necesarias para la correcta ejecución de su cliente, usuario y clave empleados en la base de datos y detalle del nombre y versión del sistema de gestión de base de datos utilizado. Si se han utilizado ficheros, incluir el (los) fichero(s) en el/los que se almacenan los datos, etc.

3. **WAR con el código fuente**[7] generado en Eclipse con el código del servicio (WAR). Se añaden seleccionado la opción de incluir archivos con el código fuente (source files). COMPROBAD QUE EL WAR TIENE INCLUIDO EL CÓDIGO FUENTE.

4. Proyecto con el código del cliente (comprimido ZIP ó RAR)

FECHA ENTREGA: 16 abril 2023 hasta las 23:55

Criterios de evaluación:

- Diseño 60% nota final
    - Definición URIs y recursos 30%
    - Verbos, parámetros y navegabilidad, 25%
    - Códigos http 10%
    - Estructuras de datos; XML y JSON 15%
    - Memoria 20%
- Implementación servicio y cliente 40%
    - Servicio 75% y Cliente 25%

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

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