Sistemas Orientados a Servicios – 2023-24

Práctica: Diseño e implementación de un servicio web RESTful

Se trata de diseñar e implementar una API REST y un prototipo funcional de un servicio sencillo que simule una aplicación de una comunidad online para la gestión y recomendación de vinos, donde los usuarios de esta comunidad 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 pueden puntuar vinos, seguir otros usuarios o realizar búsquedas sobre vinos concretos o descubrir recomendaciones basadas en sus gustos personales. Para ello se debe tener en cuenta que los vinos deben estar representados al menos por el nombre de la botella, bodega a la que pertenece, añada de la botella, denominación u origen (país o región), tipo de vino (tinto, blanco, rosado…etc.), tipos de uva (un vino puede estar formado por varios tipo de uva en porcentaje).

El servicio debe implementar a través de la API las siguientes operaciones:

- Añadir un usuario nuevo a la comunidad. Los datos básicos de un usuario son: nombre de usuario, fecha de nacimiento para comprobar que es mayor de edad y correo electrónico. Si el usuario no es mayor de edad no podrá ser registrado en el servicio y se deberá devolver un mensaje de error.
- Ver los datos básicos del usuario.
- Cambiar datos básicos de nuestro perfil de usuario.[1]
- Borrar nuestro perfil.
- Obtener una lista de todos los usuarios en la aplicación. 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 añadir un vino a su lista y agregarle una puntuación (0 a 10).
- Un usuario puede eliminar vinos de su lista personal.
- Un usuario puede modificar la puntuación de un vino concreto de su lista.
- Obtener una lista con todos los vinos de un usuario. Esta lista debe permitir la opción de ser filtrada por fecha de adición o limitar la cantidad de información obtenida por cantidad (e.g. los X primeros, los elementos entre X e Y, etc.).
- Se debe dar la opción de realizar filtros en la lista de vinos de un usuario para poder buscar vinos de un usuario por característica/s de vinos: tipo de vino, uva, origen, año…etc, o la combinación de varios de estos filtros.
- Un usuario puede añadir seguidores entre los usuarios existentes.
- Un usuario puede eliminar un seguidor que tuviera.
- Obtener un listado de todos los seguidores de un usuario. Además, esta lista debe permitir la opción de ser filtrada por patrón de nombre y/o limitar la cantidad de información obtenida.

[1] Ciertas operaciones como cambiar datos personales de un usuario, añadir seguidores…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 y navegabilidad vistos en clase. Los resultados deben estar paginados y deben mostrar las URIs de cada recurso.

- Un usuario puede obtener la lista de vinos de un seguidor concreto, pudiendo filtrar este listado por las características de los vinos (bodega, año, origen, tipo de vino…etc.). Esta lista debe permitir la opción de ser filtrada por fecha de adición o limitar la cantidad de información obtenida por cantidad (e.g. los X primeros, los elementos entre X e Y, etc.).
- Se debe permitir consultar fácilmente un sistema de recomendaciones personalizado para un usuario. Esta operación debe devolver toda la información del usuario (datos personales), un listado con sus 5 últimos vinos añadidos, un listado con sus 5 vinos con mayor puntuación y otro listado con los 5 mejores vinos de todos sus amigos.

Se pide:

Teniendo en cuenta estos aspectos, esta práctica se plantea atendiendo a dos objetivos fundamentales:

1. 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.

2. Que el alumno sepa implementar ese servicio usando el entorno Eclipse + Tomcat y la implementación de referencia de la API JAX-RS (Jersey). Y posteriormente verifique su correcto funcionamiento con un cliente REST (Postman) y desarrolle su propio cliente en java.

Para ello se divide el trabajo a realizar en 3 categorías:

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

- Identifique primero 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 JSON necesarios para el servicio. Diseñe los esquemas 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 SWAGGER para cada recurso, método y representación. Se debe facilitar el fichero YAML para su comprobación y/o el enlace público a la API. Además, se debe incorporar en la memoria las capturas de pantalla de cada operación. Alternativamente se podrá utilizar la siguiente tabla **para cada recurso definido y método soportado** para dicho recurso. Aunque se valorará positivamente el uso de Swagger.

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/JSON + referencia al JSON Schema)	
Devuelve	200	OK +POX (tipo MIME o application/JSON + referencia al JSON Schema)
	401	Unauhtorized
	Etc.	Etc.

b) Implementación de la API REST del servicio utilizando la implementación de referencia del estándar para REST JAX-RS vista en clase o cualquiera de sus versiones (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 en la memoria el método de persistencia de datos seleccionado (base de datos relacional, no relacional, 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.

La entrega se realizará en Moodle y contendrá al menos cuatro ficheros:

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

   1. Resumen del diseño del servicio como se especifica en la página anterior (a), incluyendo las capturas de pantalla de SWAGGER[6] o tablas con el resumen de las operaciones del servicio y el diseño de la persistencia de los datos escogida (Ej, base de datos relacional o no relacional, 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 llamada a la operación como los detalles del resultado de esa llamada**. Se valorará positivamente la definición de colecciones de pruebas y su automatización. Se deben detallar en la memoria.

   3. Capturas de la ejecución del cliente de prueba para las operaciones anteriormente referidas.

[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
[6] Se deben incorporar las capturas de pantalla de cada operación de Swagger y adjuntar en la entrega el fichero YAML. Se valorará positivamente el uso de Swagger.

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 seleccionando la opción de incluir archivos con el código fuente (source files).

COMPROBAD QUE EL WAR TIENE INCLUIDO EL CÓDIGO FUENTE O LA PRACTICA NO PODRÁ SER CORREGIDA.

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

FECHA ENTREGA: 7 abril 2024 hasta las 23:55

Criterios de evaluación:

- Diseño 60% nota final
  - Definición URIs y recursos 30%
  - Verbos, códigos http, parámetros 20%
  - Estructuras de datos JSON, paginable y navegable 15%
  - Diseño en swagger y colecciones de pruebas automatizadas 10%
  - Memoria 25%
- Implementación servicio y cliente 40%
  - Servicio 75% y Cliente 25%

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