Desarrollo web en entorno servidor

Desarrollo de API

Servicios web – Tipos de servicios web – REST y RESTful
Métodos HTTP – Códigos de respuesta – JSON – Convenciones
Versionado – REST API en Spring Boot

IES Clara del Rey – Madrid

Servicios web – Qué son

Aplicación que proporciona una interfaz estándar (una API) para que distintas aplicaciones la consuman (utilicen).

Aunque hay servicios que utilizan otros protocolos, los servicios web usan HTTP y HTTPS para el envío de mensajes.

Al usar un mecanismo estándar para la comunicación, no importa el lenguaje (C#, Java, PHP, Python…) en el que se hayan programado, ni el sistema operativo (Linux, Windows, MacOs…) en el que se ejecuten.

En general, soportan alguna clase de lenguaje de marcas (o varios) para transferir la información entre el servidor (el que expone el servicio) y el cliente (el que lo consume). Lo más habitual es JSON o XML.

Servicios web – Utilidad y casos de uso

Los usos más habituales de los servicios web son:

- API para aplicaciones móviles. Las aplicaciones móviles consultan / almacenan en el servicio web los datos necesarios.
- API para aplicaciones web. Igual que las apps móviles, pero las aplicaciones basadas en JS o en frameworks JS (React, Angular, Vue...).
- Integración de sistemas. Para comunicar distintos sistemas entre sí. Por ejemplo, un CRM y un ERP. También integración con proveedores de servicios, como pasarelas de pagos.
- IoT. Los dispositivos conectados, como los sensores, suelen entregar la información a alguna clase de servicio web.

Servicios web – Tipos de servicios

- REST (Representational State Transfer)
  - Basado en principios de la arquitectura web.
  - Usa HTTP/ HTTPS y formatos ligeros como JSON o XML.
  - Más flexible y fácil de implementar que otros, como SOAP.
- SOAP (Simple Object Access Protocol)
  - Basado en XML.
  - Usa HTTP/HTTPS o SMTP.
  - Más estricto en su estructura y seguridad.
  - Muy común hace algún tiempo, va cediendo terreno frente a REST.

Servicios web – Tipos de servicios

- GraphQL (Graph Query Language)
  - Desarrollado originalmente por Facebook.
  - Permite más flexibilidad, consultando solo los datos necesarios.
  - Más eficiente en el uso de ancho de banda.
  - Útil en aplicaciones que requieren respuestas optimizadas y personalizadas, con muchas variaciones en su estructura.
- RPC (Remote Procedure Call)
  - Permite llamar funciones remotas directamente.
  - Puede usar JSON-RPC o XML-RPC.
  - Usado en sistemas embebidos o comunicación entre microservicios.

REST API – ¿Qué es?

REST (Representional State Transfer) propone diseñar los servicios en función de recursos, y usando características estándares de HTTP.

En un servicio REST:

- Cada recurso se representa con una URL única.
- Las operaciones sobre recursos utilizan métodos HTTP estándar.

Ejemplos de peticiones REST de una API para gestión de usuarios:

GET /usuarios – Obtener todos los usuarios
GET /usuarios/{id} – Obtener un usuario por ID
POST /usuarios – Crear un nuevo usuario
PUT /usuarios/{id} – Actualizar un usuario existente
DELETE /usuarios/{id} – Eliminar un usuario

REST API – Principios fundamentales

- Cliente/Servidor – Cliente: consume el servicio. Servidor: lo ofrece.
- Sin estado (Stateless) – Al estar basado en HTTP, que es un protocolo sin estado, un servicio REST también lo es. Si se desea mantener un estado hay que utilizar algún sistema de sesiones o similar
- Cacheable – El servidor puede indicar indican si se pueden almacenar datos en caché para mejorar rendimiento. Con cabeceras en la respuesta HTTP, como “Cache-Control: max-age=3600, public”
- Interfaz uniforme – Todas las interacciones siguen reglas claras, usan métodos HTTP estándar y formatos de datos conocidos (JSON o XML).

REST API vs RESTful API

Una API REST no es exactamente igual que una API RESTful.

Una API RESTful sigue estrictamente los principios REST:

- Uso de URI para para identificar los recursos.
- Uso de los métodos HTTP adecuados en cada caso.
- Concibe el cacheo como parte inherente a la API.
- Son más estables y predecibles: no se “salen” de los estándares.
- Evitan integrar otros mecanismos como RPC.
- Siempre sin estado.

Las API REST pueden salirse un poco de estos principios.

REST API – Métodos HTTP

En REST, los recursos se manejan con un método HTTP específico para cada tipo de operación:

- GET Obtener un recurso o lista de recursos
- POST Crear un nuevo recurso
- PUT Actualizar completamente un recurso existente
- PATCH Modificar parcialmente un recurso
- DELETE Eliminar un recurso

No sería correcto borrar un recurso con un método POST, o pedir un listado de elementos con un PUT.

REST API – Códigos de respuesta

El servidor debe responder usando códigos estándares HTTP:

Rango	Categoría	Uso
1xx	Informativos	La solicitud ha sido recibida y el servidor continúa con el procesamiento.
2xx	Éxito	La solicitud se ha procesado correctamente.
3xx	Redirección	El cliente debe realizar una petición adicional para completar la solicitud.
4xx	Error de cliente	Hubo un problema con la solicitud del cliente, no es una petición correcta por algún motivo.
5xx	Error de servidor	El servidor ha fallado al procesar la solicitud. La solicitud era correcta, pero hubo un fallo al procesarlo.

REST API – Códigos de respuesta habituales

Código	Nombre	Uso
200	Ok	Solicitud procesada con éxito.
201	Created	Se creó un nuevo recurso correctamente.
400	Bad request	La petición es incorrecta o tiene datos no válidos
401	Unauthorized	El cliente no se ha autenticado.
403	Forbidden	Cliente autenticado, pero no tiene permisos para el recurso.
404	Not found	El recurso no existe
409	Conflict	Hay un conflicto con el estado actual del recurso
500	Internal server error	Error interno al procesar la petición. La petición era correcta, pero se ha producido un error.
503	Service unavailable	El servicio no está disponible.

REST API – JSON

Aunque pueden usar otros, JSON (JavaScript Object Notation) es el formato más común en la comunicación entre cliente y servidor en servicios REST.

Es un formato ligero para intercambio de datos. Se basa en la sintaxis de objetos de JavaScript, pero es compatible con la mayoría de los lenguajes de programación, bien directamente, bien con uso de clases específicas.

- Ligero y eficiente: Consume menos ancho de banda que XML.
- Fácil de leer y escribir: Sintaxis simple basada en pares clave-valor
- Independiente del lenguaje: Compatible con casi todos los lenguajes
- Soporte para estructuras anidadas: Permite listas y objetos dentro de otros objetos.

REST API – JSON – Tipos de datos

Un JSON puede contener los siguientes tipos de datos:

Tipo	Ejemplo
Texto	“nombre”: “Juan”
Números enteros o decimales	“importe”: 345.23
Boolean (true / false)	“completado”: true
Arrays, listas colecciones	“numeros”: [1, 4, 6, 9]
Objetos	“persona”: { nombre: “Juan”, “edad”: 40 }
Null	“teléfono”: null

La clave siempre va entre comillas. El valor puede ir o no en función del tipo de dato. Si se quieren incluir binarios en un JSON deben codificarse con Base64 o similar.

REST API – Convenciones

A la hora de diseñar las URL hay una serie de convenciones que se recomienda seguir (aunque puede haber excepciones):

- Los recursos se nombran en plural:
  - Mejor: /api/usuarios
  - Peor: /api/usuario
  - Excepciones – URL para elementos únicos en al API:
    - Autorización: /api/auth
    - Carro de compra en tienda: /api/cart

REST API – Convenciones

A la hora de diseñar las URL hay una serie de convenciones que se recomienda seguir (aunque puede haber excepciones):

- Uso de sustantivos, no de verbos. Los verbos son los métodos HTTP:
  - Mejor: POST /api/usuarios - Crear usuario
  - Peor: POST /api/usuarios/crear - No hace falta el crear
- Anidamiento de recursos:
  - Mejor: /api/usuarios/3/pedidos - Pedidos del usuario 3
  - Peor: /api/pedidos?id-usuario=3

REST API – Convenciones

A la hora de diseñar las URL hay una serie de convenciones que se recomienda seguir (aunque puede haber excepciones):

- Filtros, orden y paginación en parámetros de la query:
  - Mejor: /api/productos?cat=electrónica
  - Peor: /api/productos/cat/electrónica
  - Mejor: /api/productos?cat=electrónica&pag=1&sort=nombre
  - Peor: /api/productos/cat/electrónica/1/nombre

REST API – Versionado

Como las API dan soporte a sistemas ajenos a ellas, es difícil sincronizar actualizaciones de la API y actualizaciones de las aplicaciones que las consumen.

El ejemplo más evidente es las aplicaciones móviles. No se puede garantizar que los usuarios actualicen las APPS a la vez que las API, y no se puede retirar alegremente de servicio una versión de API que pueden usar muchos usuarios.

Para esto la API se versiona. Esto es, cuando se libera una nueva versión se mantienen en servicio una serie de versiones anteriores.

Las dos estrategias más utilizadas para el versionado de versiones son el versionado por URL y el versionado por cabecera.

REST API – Versionado

Versionado por URL: se incluye la versión en la URL, dentro del path:

- /api/v1/productos
- /api/v1/productos

Versionado por cabecera: el cliente incluye en la petición una cabecera con la versión de API. Si se incluye se usa una versión por defecto.

Se puede usar una cabecera Accept personalizada:

- Accept: application/vnd.miapi.v1+json
- Accept: application/vnd.miapi.v1+json

No usar versión, incluso si no se necesita, puede ser un problema en el futuro, cuando se necesite.

REST API – Versionado

En el versionado por cabecera hay una convención para el formato de la cabecera. Se recomienda usar el versionado MIME type vendor-specific.

Ejemplo:

- Accept: application/vnd.miapi.v1+json

La cabecera se divide en:

- application: indica que es un tipo de contenido de aplicación
- vnd: indica que es de un "vendor", una aplicación "no estandarizada"
- miapi: identifica la aplicación del "vendor"
- v1: versión de la API
- +json: indica que la respuesta se prefiere en formato json

REST API en Spring Boot

Se utiliza el starter “Spring Web”. La dependencia necesaria es:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Spring Web proporciona dos anotaciones para controladores:

- @Controller – Para desarrollo de páginas web con MVC + motor de plantillas (Thymeleaf, JSP, JTE)
- @RestController – Para desarrollo de API REST

@RestController es una extensión de @Controller. Igual, pero añade @ResponseBody, que es la anotación encargada de devolver JSON

REST API en Spring Boot
@Controller vs @RestController

Característica	@Controller	@RestController
Propósito	Desrrollo web MVC con modelos, plantillas y HTML	Desarrollo de servicios REST, que devuelven JSON
Los métodos devuelven por defecto	Vistas (thymeleaf, jsp)	JSON o XML
Uso de @ResponseBody	Se necesita en los métodos que tienen que devolver JSON	No se necesita @ResponseBody en ningún método
Integración con plantillas	Se usan plantillas para dar “forma” al modelo	No se usan plantillas, se devuelven datos, objetos

REST API en Spring Boot
Mappings, parámetros, wildcards, etc.

En lo que respecta a los mapeos de las URL a los métodos de los controladores, todo es igual que con Spring Web MVC. Todos los elementos funcionan de la misma forma:

- @RequestMapping (y sus “variantes” @GetMapping, @PostMapping).
  - Se añaden @PutMapping, @PatchMapping y @DeleteMapping.
- @PathVariable y @RequestParam
- Uso de * y ** para realizar comodines en las rutas.
- Restricciones usando expresiones regulares, o restricciones por cabecera o por parámetros.

REST API en Spring Boot – Versionado

Para implementar el versionado por url en una Spring API simplemente hay que usar RequestMapping con el path adecuado:

```
@RestController
@RequestMapping("/api/v1/usuarios")
public class V1UsuarioController { ... }

@RestController
@RequestMapping("/api/v2/usuarios")
public class V2UsuarioController { ... }
```

Se necesitarán varios controladores, uno por cada versión de la API.

Si se quiere minimizar código duplicado se puede usar herencia para unificar en un controlador "BaseUsuarioController" el código común a todas las versiones.

REST API en Spring Boot – Versionado

Para implementar el versionado por cabecera, se usa el parámetro "headers" en los mappings. Se puede poner método a método o aplicarlo en todo el controlador:

```
@RestController
@RequestMapping(value = "/api/productos",
                headers = "Accept=application/vnd.miapi.v1+json")
public class V1UsuarioController { ... }

@RestController
@RequestMapping(value = "/api/productos",
                headers = "Accept=application/vnd.miapi.v2+json")
public class V1UsuarioController { ... }
```

También se puede usar herencia para compartir la base de código entre clases.