Verbos HTTP y Buenas Prácticas de Diseño de API REST

Pregunta:

Explique cuáles son los verbos HTTP, las versiones HTTP, las palabras reservadas, y las buenas prácticas para diseñar una API RESTful.

Respuesta:

1. Versiones de HTTP y sus Métodos

HTTP/1.0 (1996)

Es la primera versión completa y estándar del protocolo HTTP. En esta versión, el uso de conexiones es de tipo "conexión por solicitud", lo que significa que una conexión TCP se cierra después de cada solicitud/respuesta.

Métodos (Palabras reservadas):

- **GET**: Recuperar información de un recurso específico.
- **POST**: Enviar datos al servidor para procesar (crear o modificar recursos).
- **HEAD**: Similar a `GET`, pero solo recupera los encabezados (sin cuerpo).

HTTP/1.1 (1997)

Introdujo conexiones persistentes (mantiene la conexión abierta para múltiples solicitudes/respuestas), soporte mejorado para caché y validaciones condicionales.

Métodos adicionales:

- **PUT**: Reemplazar por completo el recurso en el servidor.
- **DELETE**: Eliminar un recurso específico.
- **OPTIONS**: Describe las opciones de comunicación para el recurso.
- **TRACE**: Realiza un loopback de prueba con la solicitud a lo largo de la ruta hasta el servidor.
- **PATCH**: Aplicar modificaciones parciales a un recurso.

HTTP/2 (2015)

Mejoras significativas en el rendimiento, incluyendo la multiplexación de solicitudes/respuestas, compresión de encabezados y servidores push.

Métodos:

- Los mismos que HTTP/1.1.

HTTP/3 (Actualmente en adopción)

Basado en el protocolo QUIC, mejora la latencia y la fiabilidad.

Métodos:

- Los mismos que HTTP/1.1 y HTTP/2.

2. Lista de Códigos de Estado

1. **1xx Informational (Información):**
    - **100 Continue**: El cliente puede continuar con la solicitud.
    - **101 Switching Protocols**: El servidor acepta cambiar el protocolo de comunicación.
2. **2xx Success (Éxito):**
    - **200 OK**: La solicitud ha sido procesada con éxito.
    - **201 Created**: El recurso ha sido creado satisfactoriamente.
    - **202 Accepted**: La solicitud ha sido aceptada pero no completada.
    - **204 No Content**: La solicitud fue exitosa, pero no hay contenido en la respuesta.
3. **3xx Redirection (Redirección):**
    - **301 Moved Permanently**: El recurso ha sido movido permanentemente.
    - **302 Found**: El recurso ha sido encontrado temporalmente en una URL diferente.
    - **304 Not Modified**: El recurso no ha sido modificado.
4. **4xx Client Error (Errores del cliente):**
    - **400 Bad Request**: La solicitud tiene errores de sintaxis.
    - **401 Unauthorized**: Se requiere autenticación.
    - **403 Forbidden**: El acceso al recurso está prohibido.
    - **404 Not Found**: El recurso no fue encontrado.
    - **409 Conflict**: Conflicto en el estado del recurso.
    - **422 Unprocessable Entity**: La solicitud no puede procesarse.
5. **5xx Server Error (Errores del servidor):**
    - **500 Internal Server Error**: Error del servidor al procesar la solicitud.
    - **501 Not Implemented**: El servidor no soporta la funcionalidad requerida.
    - **502 Bad Gateway**: El servidor recibió una respuesta inválida del servidor aguas arriba.
    - **503 Service Unavailable**: El servidor no está disponible (mantenimiento o sobrecarga).
    - **504 Gateway Timeout**: El servidor no recibió respuesta a tiempo.

3. Buenas Prácticas para el Diseño de API RESTful

3.1. Usar Verbos HTTP Correctamente

- **GET**: Para leer información (no debe cambiar el estado del servidor).
- **POST**: Para crear nuevos recursos.
- **PUT**: Para actualizar o reemplazar un recurso existente.
- **DELETE**: Para eliminar un recurso.
- **PATCH**: Para modificaciones parciales en un recurso.

3.2. No Incluir Verbos en la URL

La acción (crear, actualizar, borrar) ya está implícita en el verbo HTTP, por lo que no es necesario incluir palabras como `create`, `delete` o `update` en el endpoint.

Ejemplo:

- **Correcto**:
  - `GET /libros`
  - `POST /libros`
  - `PUT /libros/:id`
  - `DELETE /libros/:id`
- **Incorrecto**:
  - `POST /createBook`
  - `GET /getBook`
  - `DELETE /deleteBook`

3.3. Usar Nombres de Recursos en Plural

Los recursos deben representarse en plural en la URL, ya que una API generalmente está gestionando una colección de recursos.

Ejemplo:

- `GET /libros` → Devuelve una lista de libros.
- `POST /libros` → Crea un nuevo libro.

3.4. Versionar la API

Es importante incluir el número de versión de la API para evitar romper la funcionalidad a usuarios existentes cuando se lancen actualizaciones.

- **Versionamiento en la URL**: `https://api.tuapp.com/v1/libros`
- **Versionamiento en encabezados**:

```
GET /libros
Headers:
  Accept: application/vnd.tuapp.v1+json
```

3.5. Manejar los Códigos de Estado Correctamente

El servidor debe responder con el código de estado HTTP adecuado para reflejar el resultado de la solicitud:

- **201 Created**: Después de crear un recurso con éxito.
- **204 No Content**: Después de eliminar un recurso.
- **404 Not Found**: Si el recurso solicitado no existe.
- **400 Bad Request**: Si hay un error en la solicitud del cliente.

3.6. Usar Filtros, Paginación y Búsquedas

Para colecciones grandes de recursos, permite al usuario filtrar resultados o solicitar páginas específicas de los datos:

- **Filtros**: `GET /libros?autor=JohnDoe`
- **Paginación**: `GET /libros?page=2&limit=10`

3.7. Documentar la API

Siempre proporciona una documentación clara y detallada sobre cómo interactuar con la API, las solicitudes esperadas, los métodos, las respuestas y los códigos de error.

Resumen de Buenas Prácticas de API REST:

1. **Usa verbos HTTP correctamente**.
2. **Evita incluir verbos en la URL**, ya que los métodos HTTP definen la acción.
3. **Usa nombres de recursos en plural**.
4. **Versiona la API** correctamente, de preferencia en la URL.
5. **Maneja los códigos de estado HTTP correctamente** en cada respuesta.
6. **Usa filtros y paginación** en colecciones grandes.
7. **Documenta la API** para los desarrolladores que la usarán.

Estas prácticas aseguran que tu API sea fácil de entender, consistente, escalable y mantenible.