M8_AE1_ABP-Ejercicio individual[Actividad Evaluada]

1. ¿Qué Significa REST y Para Qué Sirve?

REST significa **Representational State Transfer** (Transferencia de Estado Representacional).

- **Tipo de Sistemas:** Se utiliza principalmente para diseñar **servicios web** o **APIs** (Application Programming Interfaces).
- **Propósito Principal:** Servir como un **estilo de arquitectura** que define un conjunto de restricciones para construir sistemas distribuidos, escalables y sin estado, que utilizan el protocolo **HTTP** como su base de comunicación.

---

2. Características Principales de una API RESTful

Una arquitectura se considera **RESTful** si adhiere a estas restricciones clave:

1. **Separación Cliente-Servidor:** El cliente (quien solicita los datos) y el servidor (quien provee los datos) deben actuar de forma independiente, permitiendo que evolucionen por separado.
2. **Sin Estado (Stateless):** El servidor **no almacena información de sesión** del cliente entre peticiones. Cada solicitud debe contener toda la información necesaria para ser completada.
3. **Interfaz Uniforme:** Es la restricción central que simplifica la interacción. Los recursos son identificados mediante **URIs** y las interacciones se realizan utilizando un conjunto estándar de métodos **HTTP** (GET, POST, etc.).
4. **Sistema de Capas:** La arquitectura permite que existan **intermediarios** (como *proxies* o *caches*) entre el cliente y el servidor, mejorando la seguridad y la escalabilidad.
5. **Cacheable:** Las respuestas a las solicitudes pueden declararse como *cacheables* o *no cacheables* para evitar repetir peticiones y mejorar el rendimiento.

---

3. Ventajas y Desventajas de Usar REST

✅ Ventajas

1. **Escalabilidad:** Al ser *sin estado*, la carga se puede distribuir fácilmente entre múltiples servidores, facilitando el escalado horizontal.
2. **Simplicidad:** Se basa en el protocolo HTTP, ampliamente conocido, lo que hace que su desarrollo, prueba y consumo sean relativamente sencillos.

3. **Portabilidad:** La separación cliente-servidor permite que la interfaz de usuario (el cliente) se desarrolle en cualquier tecnología y se ejecute en cualquier plataforma.

❌ Desventajas

1. **Sin Estado (Stateless):** Para aplicaciones que requieren mantener un estado de sesión (como carritos de compra complejos), la necesidad de incluir la información de estado en cada petición puede aumentar la sobrecarga (overhead).
2. **Falta de Estándar Formal:** A diferencia de SOAP, REST no tiene un estándar formal estricto, lo que puede llevar a inconsistencias en la implementación si no se siguen buenas prácticas de diseño.
3. **Ineficiencia en Operaciones Múltiples:** Para obtener datos de múltiples recursos relacionados, el cliente a menudo tiene que realizar varias peticiones HTTP separadas (lo que se conoce como *over-fetching* o *under-fetching*).

---

4. ¿Qué es la Interfaz Uniforme con Mensajes Descriptivos?

La **Interfaz Uniforme** es el principio que estandariza la comunicación entre el cliente y el servidor. Es importante porque:

- **Simplicidad:** Al usar un conjunto fijo de métodos HTTP, el cliente no necesita conocer la lógica interna del servidor.
- **Independencia:** Permite la independencia tecnológica del cliente.

Los **mensajes descriptivos** (o *Self-Descriptive Messages*) complementan esto al asegurar que cada mensaje (tanto la solicitud como la respuesta) contiene suficiente información para ser procesado por el receptor. El servidor utiliza:

- **URIs** para identificar los recursos (e.g., `/productos/123`).
- **Códigos de estado HTTP** (e.g., 200, 404) para comunicar el resultado de la operación.
- **Mime Types** (e.g., `application/json`) para describir el formato de la representación de los datos.

---

5. ¿Qué Significa que las Peticiones Sean “Sin Estado”?

Que las peticiones sean **"sin estado"** (Stateless) implica que **cada solicitud HTTP es independiente y autónoma**.

- **Implicación:** El servidor **no guarda información de sesión** del cliente (como un ID de sesión) ni depende de datos de peticiones anteriores. Si el cliente necesita autenticarse o proporcionar algún contexto, debe incluir esa información (e.g., un token de autenticación) en **cada petición**.

- **Ventaja:** Esto facilita la **escalabilidad** porque permite que cualquier servidor disponible en un *pool* pueda manejar la solicitud, sin importar cuál la haya manejado previamente.

---

6. ¿A Qué Nos Referimos con Separación Entre Cliente y Servidor?

La **Separación Cliente-Servidor** significa que las preocupaciones del cliente (interfaz de usuario, experiencia) están separadas de las preocupaciones del servidor (almacenamiento de datos, lógica de negocio).

- **Promoción:** REST promueve esto porque la **interfaz uniforme** (p.ej., el uso de JSON para la comunicación) actúa como un contrato rígido que obliga a esta separación.
- **Importancia para la Escalabilidad:** Permite que los equipos de desarrollo del cliente y del servidor trabajen de forma **independiente**. El servidor puede escalar y mejorar su almacenamiento o lógica sin que el cliente necesite ser modificado, siempre y cuando se respete el contrato de la API.

---

7. ¿Qué es el Sistema de Capas en una Arquitectura REST?

El **Sistema de Capas** es una restricción que permite a un cliente conectarse al servidor final a través de **capas intermedias** sin que el cliente se dé cuenta (o se preocupe) de si está conectado directamente o no.

- **Funcionamiento:** Las capas intermedias (*caches*, *gateways*, *proxies*, balanceadores de carga) pueden interceptar la solicitud o la respuesta. Por ejemplo, un *proxy* podría encargarse de la **seguridad** (autenticación) o un *cache* podría devolver una respuesta rápidamente sin molestar al servidor final.
- **Beneficio:** Aumenta la **seguridad** (al ocultar la lógica de negocio detrás de múltiples capas) y la **eficiencia** (al permitir el *caching*).

---

8. ¿Por Qué es Importante el Versionamiento de una API?

El **Versionamiento** es crucial porque permite que los desarrolladores realicen **cambios no compatibles** (*breaking changes*) en la API (p.ej., cambiar el nombre de un campo, eliminar un *endpoint*) sin romper las aplicaciones existentes que la están consumiendo.

- **Forma 1: Versionado por URI:** Es la forma más común y clara. La versión se incluye en el camino del recurso.
    - Ejemplo: https://api.ejemplo.com/v1/usuarios

- **Forma 2: Versionado por Encabezado (Header):** La versión se especifica en un encabezado HTTP, como `Accept` o uno personalizado.
    - Ejemplo: `Accept: application/json; version=2`

---

9. Principales Verbos HTTP Usados en REST

Los verbos HTTP se utilizan para realizar operaciones CRUD sobre los recursos:

- **GET: Obtener/Leer** un recurso o una lista de recursos. (Es **idempotente** y **seguro**).
    - Ejemplo: `GET /usuarios`
- **POST: Crear** un nuevo recurso.
    - Ejemplo: `POST /usuarios`
- **PUT: Actualizar** un recurso, generalmente **reemplazándolo completamente** con la nueva información. (Es **idempotente**).
    - Ejemplo: `PUT /usuarios/456`
- **PATCH: Actualizar** un recurso, aplicando una **modificación parcial** a los datos.
    - Ejemplo: `PATCH /usuarios/456`
- **DELETE: Eliminar** un recurso específico. (Es **idempotente**).
    - Ejemplo: `DELETE /usuarios/456`

---

10. ¿Cómo se Deben Nombrar los Recursos en una API RESTful?

La buena práctica principal es nombrar los recursos usando **sustantivos en plural**.

- **Recomendación:** Usar `POST /usuarios` para crear un usuario, `GET /usuarios` para obtener la lista de todos y `GET /usuarios/{id}` para obtener uno en particular.
- **Razón:** En REST, el recurso representa una **colección** de entidades, y las operaciones (verbos HTTP) definen la acción. Evitar el uso de verbos en el URI (`/obtenerUsuario`) promueve la **Interfaz Uniforme** al dejar que el verbo HTTP (`GET`) defina la acción.

---

11. ¿Qué son los Códigos de Estado HTTP y Cómo se Deben Usar?

Los **Códigos de Estado HTTP** son números de tres dígitos que el servidor envía en la respuesta para indicar el resultado del intento del cliente de realizar una solicitud.

- **Uso:** Son esenciales para la **Interfaz Uniforme** ya que comunican el estado de la manera más clara y estandarizada posible, sin obligar al cliente a analizar el cuerpo de la respuesta para saber si hubo un error.

Ejemplos Comunes:

1. **200 OK (Éxito):** La solicitud fue exitosa. Comúnmente usado para **GET** o **PUT/PATCH** exitosos.
2. **404 Not Found (Error del Cliente):** El recurso solicitado no existe en el servidor.
3. **500 Internal Server Error (Error del Servidor):** Un error genérico que indica que algo salió mal en el servidor al intentar procesar la solicitud.

---

12. ¿Cuál es el Formato de Salida Más Común en una API RESTful?

El formato de salida más común y recomendado es **JSON** (JavaScript Object Notation).

- **Ventajas:** Es ligero, fácil de leer y escribir (tanto para humanos como para máquinas), y se integra perfectamente con JavaScript y la mayoría de los lenguajes de programación modernos.
- **Alternativa: XML** (Extensible Markup Language) también se puede usar, pero ha sido ampliamente reemplazado por JSON en la mayoría de las APIs modernas debido a la verbosidad de XML.

---

13. ¿Qué es Búsqueda y Filtrado en el Contexto de REST?

La **Búsqueda y el Filtrado** se refieren a la capacidad que tiene el cliente de limitar o refinar el conjunto de datos que se le devuelve de un recurso a través de la URI.

- **Mecanismo:** Esto se realiza usando **Parámetros de Consulta** (*Query Parameters*) que se adjuntan al final del URI después de un signo de interrogación `?`.
- **Ejemplo:** Para el recurso `/productos`, el cliente puede solicitar solo los productos de una categoría específica o que coincidan con un término de búsqueda:
    - `GET /productos?categoria=ropa&precioMax=50`
- **Uso:** Estos parámetros no cambian la identidad del recurso (`/productos`), solo cómo se filtra su representación.

---

14. ¿Qué es HATEOAS y Qué Valor Aporta a una API RESTful?

HATEOAS (Hypermedia As The Engine Of Application State) es la restricción más avanzada y a menudo la menos implementada de REST.

- **Concepto:** Significa que las respuestas de la API no solo deben incluir los datos solicitados, sino también **enlaces de hipermedia** (*links*) que el cliente puede usar para realizar las siguientes acciones posibles.
- **Valor Aportado:** Convierte la API en una "máquina de estados". El cliente no necesita codificar las URIs, sino que simplemente sigue los enlaces que el servidor proporciona. Esto mejora la **descubribilidad** y la **capacidad de evolución** de la

API, ya que los *endpoints* pueden cambiar sin que el cliente tenga que actualizar su lógica, siempre que siga los nombres de los enlaces.