**APIs – Clase 10**

En esta clase tendrá la oportunidad de aprender las características de las API REST, API First, Restul, XML y Json.

**API FIRST**

La irrupción de nuevos dispositivos e internet de las cosas, ha reemplazado la idea de hacer una plataforma web para convertirse en un plataforma que pueda estar presente, utilizada, en la mayor cantidad de dispositivos presentes: móviles, tabletas, TV y en cualquier cosa que esté conectado a interrnet.

La estrategia API First debe empezar con la idea de definir la API internamente antes que cualquier otro desarrollo. Ya sea ese proyecto una *web* o una *app* móvil, o ambas.

Deben fijarse los recursos que van a ser utilizados previamente. Antes, la API era creada después que las WEBs fueran creadas para interactuar con ellas.

Eso representaba situaciones complicadas al no haber sido optimizadas para tal uso, además de una importante carga de *legacy code*.

**API REST**

Este nuevo enfoque de desarrollo de proyectos y servicios web fue definido por **Roy Fielding**, quien también realizó la especificación HTTP.

En la actualidad no existe proyecto o aplicación que no disponga de una API REST para la creación de servicios profesionales.

Las redes sociales y muchas empresas generan negocios gracias a REST y las API REST.

Sin ellas, todo el crecimiento en horizontal sería prácticamente imposible. Esto es así porque REST es el estándar más lógico, eficiente y habitual en la creación de API para servicios de Internet.

REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

Es una alternativa en auge a otros protocolos estándar de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también mucha complejidad.

A veces es preferible **una solución más sencilla de manipulación de datos como REST**.

Es un p**rotocolo cliente/servidor sin estado**: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla.

Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como **protocolo cliente-caché-servidor sin estado**: existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un futuro **la misma respuesta para peticiones idénticas**. De todas formas, que exista la posibilidad no significa que sea lo más recomendable.

Un servicio sin estado genera una respuesta que enlaza al número de la siguiente página del conjunto y deja que el cliente haga lo que tenga que hacer para conservar ese valor.

Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro:

**POST** (crear), **GET**(leer y consultar), **PUT**(editar) y **DELETE** (eliminar).

**Los objetos en REST siempre se manipulan a partir de la URI**.

Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST.

La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.

**Interfaz uniforme**: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.

**Sistema de capas**: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.

**Uso de hipermedios**: es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Este principio es el que define que cada vez que se hace una petición al servidor y éste devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.

**VENTAJAS QUE OFRECE REST PARA EL DESARROLLO.**

**Separación entre el cliente y el servidor**: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos.

Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.

**Visibilidad, fiabilidad y escalabilidad**. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas.

Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta.

Esta separación facilita tener en servidores distintos el *front* y el *back* y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

**La API REST siempre es independiente del tipo de plataformas o lenguajes**: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo

Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

**JSON**

Es un formato de datos basado en texto que sigue la sintaxis de objeto de JavaScript, popularizado por Douglas Crockford. Puede ser utilizado independientemente de JavaScript

Muchos ambientes de programación poseen la capacidad de leer (analizar; parse) y generar JSON. Convertir una cadena a un objeto nativo se denomina *parsing*, convertir un objeto nativo a una cadena para que pueda ser transferido a través de la red se denomina *stringification*

JSON es una cadena cuyo formato recuerda al de los objetos literales JavaScript.

Es posible incluir los mismos tipos de datos básicos dentro de un JSON que en un objeto estándar de JavaScript - cadenas, números, arreglos, booleanos, y otros literales de objeto. Esto permite construir una jerarquía de datos.



Se pueden insertar espacios en blanco entre cualquier par de tokens. Exceptuando algunos detalles de codificación, eso describe completamente el idioma.

Los JSON son cadenas - útiles cuando se quiere transmitir datos a través de una red. Debe ser convertido a un objeto nativo de JavaScript cuando se requiera acceder a sus datos. JavaScript posee un objeto global [JSON](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON) que tiene los métodos disponibles para convertir entre ellos.

Un objeto JSON puede ser almacenado en su propio archive: un archivo de texto con una extension .json, y una [MIME type](https://developer.mozilla.org/es/docs/Glossary/MIME_type) de application/json. JSON es sólo un formato de datos — contiene sólo propiedades, no métodos.

JSON requiere usar comillas dobles para las cadenas y los nombres de propiedades. Las comillas simples no son válidas. Una coma o dos puntos mal ubicados pueden producir que un archivo JSON no funcione.

Es posible validar JSON utilizando una aplicación como [JSONLint](http://jsonlint.com/).

JSON Puede tomar la forma de cualquier tipo de datos que sea válido para ser incluido en un JSON, no sólo arreglos u objetos

Una cadena o un número único podrían ser objetos JSON válidos.

En JSON, sólo las cadenas entre comillas pueden ser utilizadas como propiedades.

**Ejercicios:**

1. **Desarrolle ventajas y desventajas de API REST**

**Ventajas:**

1. Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
2. Visibilidad, fiabilidad y escalabilidad. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el *front* y el *back* y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
3. La API REST siempre es independiente del tipo de plataformas o lenguajes: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo

**Desventajas:**

1. Mayor complejidad en la infraestructura: Es necesario diseñar sistemas para trabajar sin estado, lo que requiere gestionar tokens y otros mecanismos de autenticación.
2. Distribución del sistema: En REST, diferentes servidores pueden no estar sincronizados, complicando la gestión de sesiones y datos.
3. Tiempos iniciales de desarrollo más largos: Configurar y desarrollar la API requiere más tiempo al inicio, especialmente si la API debe ser consumida por múltiples sistemas.
4. Rigidez en el desarrollo: La falta de sincronización entre el backend REST y los frontales puede generar retrasos si el cliente necesita funcionalidades que aún no están listas en el servidor.
5. Mayor curva de aprendizaje: Además de lenguajes y librerías habituales, se necesita conocimiento profundo del protocolo HTTP y sus particularidades.
6. Cambio de mentalidad: Requiere abandonar la noción tradicional de tener todos los recursos centralizados en un solo servidor. Esto puede resultar desafiante para los equipos de desarrollo acostumbrados a enfoques tradicionales.
7. **¿En qué casos se utiliza el REST API?**

**Su respuesta**

* **Interfaz uniforme**: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
* **Sistema de capas**: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
* **Uso de hipermedios**: es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario puede navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos. Este principio es el que define que cada vez que se hace una petición al servidor y éste devuelve una respuesta, parte de la información que contendrá serán los hipervínculos de navegación asociada a otros recursos del cliente.

1. En qué casos en propicio utilizar JSON

* **Transferencia de datos entre frontend y backend:** Por su estructura ligera y fácil de interpretar.
* **APIs RESTful:** JSON se utiliza comúnmente para enviar y recibir datos, mejorando la compatibilidad entre sistemas.
* **Almacenamiento de configuraciones y datos estructurados:** Gracias a su formato sencillo y autodescriptivo.
* **Intercambio entre sistemas heterogéneos:** Muchos lenguajes soportan JSON, lo que facilita su uso en entornos variados.