

Diseño de interfaces web

Hito individual

Sergio Felipe García

ORIGINAL

Introducción

Introducción	1
Actividad 1	2
¿Qué es una API?	2
Razones y beneficios que ofrecen las APIs.	2
Tipos de APIs:	3
API REST:	3
GET	3
POST	4
PUT	5
DELETE	6
HEAD	6
API SOAP:	7
Diferencias entre SOAP y REST	8
Diferencias entre API y SDK	9
Ejemplos de API en diferentes plataformas	10
APIs Investigadas para la construcción de la aplicación del hito.	11
Construcción de la aplicación en el marco de el API de faketoreapi	11
Conducta caja blanca y pruebas de caja negra	12
Caja blanca	12
Inicio de sesión	12
Carrito de compra	13
Comprobación login usuario para ver su perfil	13
Caja negra	14
Inicio de sesión	14

Actividad 1

¿Qué es una API?

Una API (Interfaz de Programa de Aplicación). Tal y como su nombre indica es un interfaz que se ofrece para operar con un programa. A diferencia de una Aplicación Web que normalmente ofrece un interfaz de usuario a través de un navegador, las API ofrecen a los programadores una interfaz para que un programa o un sistema pueda realizar operaciones e interactuar de manera sencilla con otro sistema o programa. En resumen un API actúa como un intermediario permitiendo que un software se comunice con otro, permitiendo el intercambio de datos y funcionalidades entre ellos. Las APIs son esenciales en el desarrollo de software, ya que permiten la integración de servicios y la creación de aplicaciones más poderosas.

Razones y beneficios que ofrecen las APIs.

Reutilización de código: Las APIs nos permiten acceder a funciones y datos preexistentes, lo que ahorra tiempo y esfuerzo al no tener que volver a escribir o recrear código desde cero.

Integración de aplicaciones: Las APIs ayudan a conectarse a diferentes aplicaciones y sistemas, lo que facilita la creación de soluciones más amplias y completas.

Escalabilidad: Las APIs permiten que las aplicaciones crezcan y evolucionen, ya que podemos agregar nuevas funciones o servicios utilizando APIs existentes sin alterar el código principal.

Flexibilidad: Las APIs son muy flexibles porque nos permiten elegir las tecnologías adecuadas para una tarea específica. Podemos utilizar diferentes APIs para diferentes funciones dentro de la aplicación que vayamos a desarrollar.

Desarrollo más rápido: Al utilizar APIs, podemos acelerar el proceso de desarrollo de software al aprovechar soluciones y servicios que ya existen en lugar de construir todo desde cero.

Acceso a datos externos: Las APIs pueden acceder a datos y recursos externos, esto nos permite trabajar con información de fuentes externas.

Reducción de costos: El uso de APIS reducirá los costos de desarrollo y mantenimiento, ya que utilizaremos servicios y recursos externos en vez de construirlo y mantenerlo todo internamente nosotros.

Tipos de APIs:

API REST:

REST está muy enfocado a las aplicaciones móviles y los servicios web.

Cuando se envía una solicitud de datos a una API de REST, se suele hacer a través de un protocolo de transferencia de hipertexto, comúnmente denominado HTTP.

Una vez que reciben la solicitud, las API REST pueden devolver mensajes en distintos formatos: HTML, XML, texto sin formato y JSON. El formato preferido es (JSON), por que, puede leerlo cualquier lenguaje de programación, es ligero y lo comprenden tanto las personas como las máquinas. De esta forma, las API de RESTful son más flexibles y se pueden configurar con mayor facilidad.

Los métodos más importantes de las API REST son: POST, GET, PUT, DELETE y HEAD.

GET

El método GET se emplea para obtener datos de un recurso concreto dependiente del API. En caso de respuesta positiva (200 OK), GET devuelve la representación en un formato concreto: HTML, XML, JSON o imágenes, JavaScript, CSS, etc. En caso de respuesta negativa devuelve 404 (not found) o 400 (bad request). Los recursos pueden ser CSS, Imágenes, datos de una BBDD etc. A continuación un ejemplo

GET	/pet/findByStatus	Finds Pets by status
-----	-------------------	----------------------

Responses

Code	Description
------	-------------

200	successful operation
-----	----------------------

Media type:

application/json

Controls Accept header.

Example Value | Schema

```
[
  {
    "id": 10,
    "name": "doggie",
    "category": {
      "id": 1,
      "name": "Dogs"
    },
    "photoUrls": [
      "string"
    ],
    "tags": [
      {
        "id": 0,
        "name": "string"
      }
    ],
    "status": "available"
  }
]
```

400	Invalid status value
-----	----------------------

POST

Aunque se puedan enviar datos a través del método GET, en muchos casos se utiliza POST. Los POST requests se envían por ejemplo con formularios:

POST /pet: Add a new pet to the store

Add a new pet to the store

Parameters Try it out

No parameters

Request body required application/json

Create a new pet in the store

Example Value | Schema

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

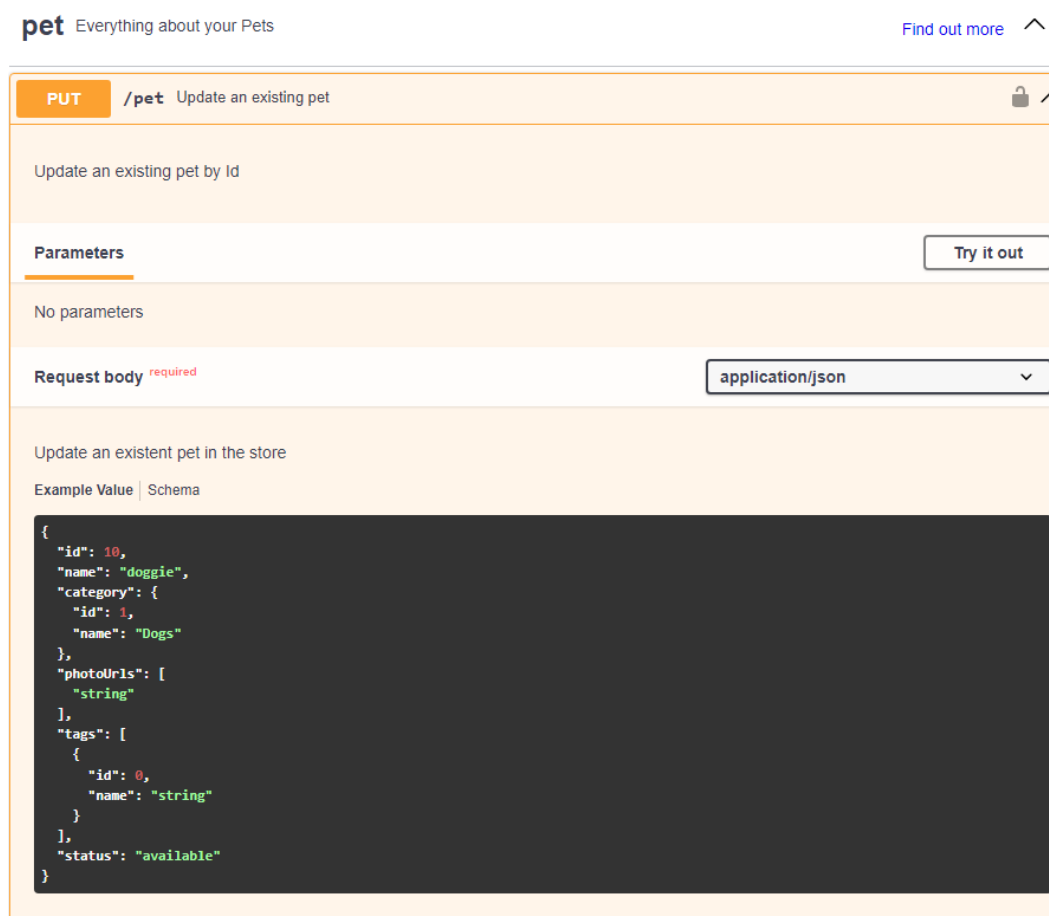
Imaginemos que tenemos un Formulario en una web y deseamos crear una nueva mascota en nuestra tienda. El desarrollador de la Web simplemente tendría que enviar los

parámetros necesarios para definir los datos de la mascota y llamar al método POST . El API (que es interfaz) recoge esos datos y se encargaría de grabarlos en la BBDD sin necesidad de que el programador tuviese que realizar conexión, manejar sentencias SQL etc etc. Por eso decimos que simplifican la labor de los desarrolladores.


El contenido va en el body del request, no aparece nada en la URL, aunque se envía en el mismo formato que con el método GET. Si se quiere enviar texto largo o cualquier tipo de archivo este es el método apropiado.

PUT

Utilizado normalmente para actualizar contenidos, pero también pueden crearlos.



pet Everything about your Pets [Find out more](#) ^

PUT **/pet** Update an existing pet  ^

Update an existing pet by Id

Parameters [Try it out](#)

No parameters

Request body required application/json v

Update an existent pet in the store

[Example Value](#) | [Schema](#)

```
{
  "id": 10,
  "name": "doggie",
  "category": {
    "id": 1,
    "name": "Dogs"
  },
  "photoUrls": [
    "string"
  ],
  "tags": [
    {
      "id": 0,
      "name": "string"
    }
  ],
  "status": "available"
}
```

DELETE

Simplemente elimina un recurso identificado en la URI. En este ejemplo se está eliminando en la URL la mascota por su id.

DELETE `/pet/{petId}` Deletes a pet

delete a pet

Parameters [Try it out](#)

Name	Description
api_key string (header)	<input type="text" value="api_key"/>
petId * required integer(\$int64) (path)	<input type="text" value="petid"/> Pet id to delete

HEAD

Es idéntico a GET, pero el servidor no devuelve el contenido en el HTTP.

HEAD /downloads/video1.mpeg HTTP/1.0	
En el encabezado que el servidor le envía de respuesta, el cliente encuentra los datos sobre el tamaño del archivo en el campo "content-length":	
age	96529
cache-control	max-age=604800
content-length	1495
content-type	text/html; charset=UTF-8
date	Fri, 06 Mar 2020 14:53:39 GMT
etag	"3147526947-gzip"
expires	Fri, 13 Mar 2020 14:53:39 GMT
last-modified	Thu, 17 Oct 2019 07:18:26 GMT
server	EC2 (dcb/7f83)
vary	Accept-Encoding
x-cache	HIT

El servidor responde a la petición HEAD con los datos clave sobre el archivo en cuestión.

API SOAP:

Este tipo de APIs imponen reglas integradas que aumentan la complejidad, lo cual puede retrasar el tiempo que tardan las páginas en cargarse. Sin embargo, cumple los estándares de cumplimiento que incluyen la seguridad, la atomicidad, la uniformidad, el aislamiento y la durabilidad (ACID), garantizando operaciones confiables de las bases de datos.

Las especificaciones comunes incluyen lo siguiente:

- Seguridad de los servicios web (WS-Security): estandariza la forma de proteger y transferir los mensajes usando identificadores únicos llamados tokens.
- Mensajería segura de los servicios web (WS-ReliableMessaging): estandariza el control de errores entre mensajes que se transfieren en infraestructuras de TI poco confiables.
- Abordaje de los servicios web (WS-Addressing): paquetes que enrutan la información como metadatos dentro de los encabezados SOAP, en lugar de mantener la información en un lugar más profundo de la red.
- Lenguaje de descripción de los servicios web (WSDL): describe qué hace un servicio web, así como dónde comienza y termina.

El envío de una solicitud de datos a una API de SOAP se puede administrar a través de cualquiera de los protocolos de la capa de la aplicación: HTTP (para los exploradores web), SMTP (para el correo electrónico), TCP, entre otros. Sin embargo, una vez que se recibe una solicitud, los mensajes SOAP de retorno deben ser documentos **XML**. Una solicitud completa a una API de SOAP no se almacena en caché por un navegador, por lo que no se puede acceder a ella después sin enviarla a la API.

Diferencias entre SOAP y REST

- Muchos sistemas antiguos utilizan SOAP, REST más nuevo, ha salido más tarde y se considera una alternativa más rápida en los escenarios basados en la Web.
- REST ofrece una implementación flexible, mientras que SOAP es un protocolo con requisitos más estrictos, como en el caso del uso de XML.
- Las API de REST son ligeras, así que son ideales para los contextos más nuevos mientras que los servicios web de SOAP ofrecen seguridad y cumplimiento de las operaciones integradas que coinciden con muchas de las necesidades empresariales, pero que también los hacen más pesados.
- Existen muchas API públicas, como la API de Google Maps (Google es una referencia mundial), por poner un ejemplo y esta usa REST.

ORIGINAL

Diferencias entre API y SDK

Ya hemos definido que era un API anteriormente, pues un SDK también están pensados para facilitar el desarrollo de aplicaciones y acelerar el desarrollo al proporcionar código y herramientas listas para usar.


Una diferencia grande, los SDKs están relacionados con una plataforma o sistema específico, lo cual limita bastante a la hora de desarrollar, esto no ocurre con las APIs pensadas para funcionar más globalmente.

Un ejemplo de un SDK conocido es el "Android SDK" (Software Development Kit) proporcionado por Google. Está diseñado para permitir a los desarrolladores crear aplicaciones para el sistema operativo Android por ejemplo el que llevan algunos teléfonos móviles. Fijémonos en lo que proporciona:

1. **Android Studio:** Un entorno de desarrollo integrado (IDE) que ofrece funciones de edición de código, depuración y diseño de interfaces de usuario.
2. **Emulador de Android:** Permite a los desarrolladores probar sus aplicaciones en un entorno simulado antes de desplegarlas en dispositivos reales.
3. **Bibliotecas de Código:** Ofrece bibliotecas de código predefinidas que facilitan el desarrollo de funciones comunes, como manipulación de imágenes, manejo de bases de datos y comunicación de red.
4. **Herramientas de Depuración:** Incluye herramientas para la identificación y corrección de errores en el código.
5. **Documentación:** Proporciona documentación detallada y ejemplos de código para guiar a los desarrolladores en el proceso de creación de aplicaciones Android.

Importante mencionar, que un SDK puede usar APIs, por ejemplo si construyes una aplicación android esta podría también interrogar al API pública de Google Maps.

La API de Google: **NO** proporciona un entorno completo de desarrollo y deja a los desarrolladores la responsabilidad de implementar la lógica de la aplicación.



En resumen, un SDK proporciona un conjunto completo de herramientas para el desarrollo de aplicaciones, mientras que una API se centra en funciones específicas y proporciona una interfaz para acceder a esos servicios.

Ambos son complementarios, y a menudo se utilizan juntos, donde un SDK puede incluir varias API para facilitar el desarrollo de aplicaciones más completas.

Ejemplos de API en diferentes plataformas

1. Google Maps API:

- Plataforma:Google
- Permite a los desarrolladores integrar mapas interactivos en sus aplicaciones y sitios web..

2. OpenWeatherMap API:

- Plataforma:OpenWeatherMap
- Ofrece datos meteorológicos actuales y pronósticos para integrar información climática en aplicaciones.

3. NASA API:

- Plataforma:NASA
- Proporciona acceso a datos relacionados con la exploración espacial, imágenes astronómicas y más.

Estas API son públicas y están disponibles para su uso.

APIs Investigadas para la construcción de la aplicación del hito.

Con el fin de desarrollar la práctica propuesta en este hito, he estado investigando diferentes APIs públicas, que proporcionan la funcionalidad necesaria para obtener datos de lo que podría ser una tienda online con los requisitos específicos que se solicitan en el hito.

A continuación muestro como referencia alguna de las URL de dichas APIs:

- <https://fakeapi.platzi.com/>
- <https://app.beeceptor.com/mock-server/fake-store-api>
- <https://docs.api2cart.com/>

Construcción de la aplicación en el marco de el API de faketoreapi

Me he decantado finalmente por <https://fakestoreapi.com/> , por varias razones:

- Primero porque es la que más fácilmente entendí en un primer momento.
- Es gratuita y se puede utilizar cada vez que necesites datos pseudo-reales para tu sitio web. No tiene limitaciones de peticiones u otro tipo de restricciones.
- Por último, al ser esta una asignatura de diseño de interfaces web, lo elegí por que permite usarla sin ejecutar ningún código en el lado del servidor.

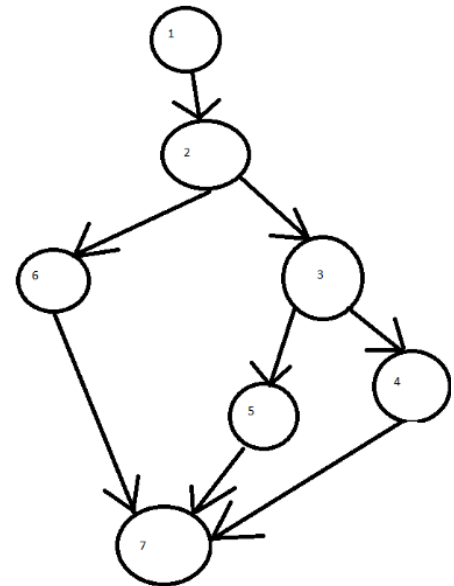
Conducta caja blanca y pruebas de caja negra

Caja blanca

Inicio de sesión

```
function iniciarSesion() {  
  // Obtiene los valores del formulario y los guarda en sus variables  
  const email_user = document.getElementById("email_user").value;  
  const password = document.getElementById("password").value;  
  
  console.log(email_user);  
  
  // Verificar si el usuario ya existe en el almacenamiento local  
  const usuariosRegistrados = JSON.parse(localStorage.getItem("usuariosRegistrados")) || [];  
  const usuarioExistente = usuariosRegistrados.find(usuario => usuario.email_user === email_user);  
  console.log(usuariosRegistrados);  
  
  if (usuarioExistente) {  
    // Comparamos que la contraseña introducida es correcta.  
    if (usuarioExistente.password === password) {  
      alert("Inicio de sesión exitoso");  
      // Guardar el email del usuario en sessionStorage  
      sessionStorage.setItem("email_usuario", email_user);  
      // Redirigir a otra página  
      window.location.href = "index.html";  
    } else {  
      alert("Contraseña incorrecta");  
    }  
  } else {  
    alert("Usuario no registrado, regístrate para continuar");  
  }  
  
  return false;  
}
```

1-2-6-7
1-2-3-5-7
1-2-3-4-7



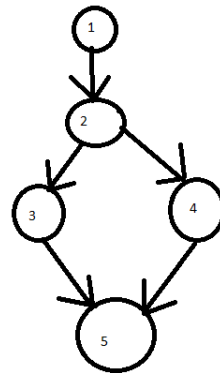
Carrito de compra

```
//Función que controla el boton clickado de agregar al carrito (nuevo codigo no tocar)
function agregarAlCarritoClickado(event){
    //compruebo que el usuario ha hecho login y tiene una sesión activa
    const emailUsuario = sessionStorage.getItem("email_usuario");
    if (emailUsuario) {
        console.log("Sesión activa para: " + emailUsuario);
        var button = event.target;
        var item = button.parentElement;
        var titulo = item.getElementsByClassName('titulo-item')[0].innerText;
        var precio = item.getElementsByClassName('precio-item')[0].innerText;
        var imagenSrc = item.getElementsByClassName('img-item')[0].src;
        console.log(imagenSrc);

        agregarItemAlCarrito(titulo, precio, imagenSrc);
        hacerVisibleCarrito();
    } else {
        // No hay sesión activa, redirigir a la página de inicio de sesión o mostrar un mensaje de error
        alert("Por favor, inicie sesión para poder comprar.");
        window.location.href = "login.html"; // Redirige a la página de inicio de sesión
    }
}
```

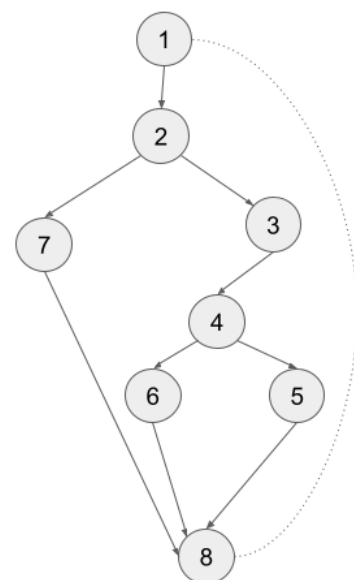
1-2-3-5

1-2-4-5



Comprobación login usuario para ver su perfil

```
document.addEventListener("DOMContentLoaded", function () {
    const emailUsuario = sessionStorage.getItem("email_usuario");
    if (emailUsuario) {
        // Puedes usar el emailUsuario para personalizar la página, cargar datos, etc.
        console.log("Sesión activa para: " + emailUsuario);
        const usuariosRegistrados = JSON.parse(localStorage.getItem("usuariosRegistrados")) || [];
        const verdatosUsuario = usuariosRegistrados.find(usuario => usuario.email_user === emailUsuario);
        if (verdatosUsuario) {
            console.log("Datos del usuario:");
            console.log("Nombre: " + verdatosUsuario.nombre);
            console.log("Apellido: " + verdatosUsuario.apellido);
            console.log("Domicilio: " + verdatosUsuario.domicilio);
            console.log("Código Postal: " + verdatosUsuario.postal);
            // Crear elementos con los datos del formulario
            var item = $('#perfil-form');
            var nombre = $('<div class="col"><label for="nombre" class="form-label">Nombre:</label><input type="text" class="form-control">');
            var apellido = $('<div class="col"><label for="apellido" class="form-label">Apellido:</label><input type="text" class="form-control">');
            var domicilio = $('<div class="col"><label for="domicilio" class="form-label">Domicilio:</label><input type="text" class="form-control">');
            var postal = $('<div class="col"><label for="codpostal" class="form-label">Código Postal:</label><input type="text" class="form-control">');
            item.append(nombre);
            item.append(apellido);
            item.append(domicilio);
            item.append(postal);
            // Añadir el formulario al contenedor
            $('#perfil-form').append(item);
            // ahora los datos de la compra
            mostrarCompras();
        } else {
            console.log("Usuario no encontrado en el almacenamiento.");
        }
    } else {
        // No hay sesión activa, redirigir a la página de inicio de sesión o mostrar un mensaje de error.
        alert("Por favor, inicie sesión para poder ver su perfil.");
        window.location.href = "login.html"; // Redirige a la página de inicio de sesión
    }
});
```



1-2-3-4-5-8

1-2-3-4-6-8

1-2-7-8

Caja negra

Inicio de sesión

Mi aplicación requiere **iniciar sesión** para poder comprar en la tienda.

Las características serían:

1. Para poder iniciar sesión el usuario ha tenido que registrarse previamente.
2. Debe introducir sus datos de usuario y contraseña
3. El programa comprueba si nombre de usuario existe “email-usuario”
4. El programa comprueba si la password es correcta.
5. Si ambas condiciones se cumplen, el programa guarda el nombre del usuario para mantener la sesión y le redirige a la página principal de la tienda.
6. Si alguna de las 2 condiciones no se cumple muestra un mensaje de error.

Condición de entrada	Clase válida	Clase inválida
email-usuario	= <code>[A-Z,a-z] >3</code> y “texto@texto” 1	email_usuario!= <code>[A-Z,a-z] >3</code> y 6 email_usuario!= “texto@texto”
email-usuario tiene que ser rellenado	email-usuario!=null 2	email-usuario=null 7
contraseña-usuario tiene que ser rellenada	contraseña-usuario!=null 3	contraseña-usuario=null 8
email-usuario tiene que existir	email-usuario=true 4	email-usuario=false 9
contraseña-usuario tiene que existir	contraseña-usuario=true 5	contraseña-usuario=false 10

Prueba	Condiciones	Resultado
sergio.felipe@campusfp.es 123	1,2,3,4,5	ok
123	3,5, 6,7	error
@pep	2, 6,8,9	error
pep@ 123	2,3, 6,9,10	error

ORIGINAL