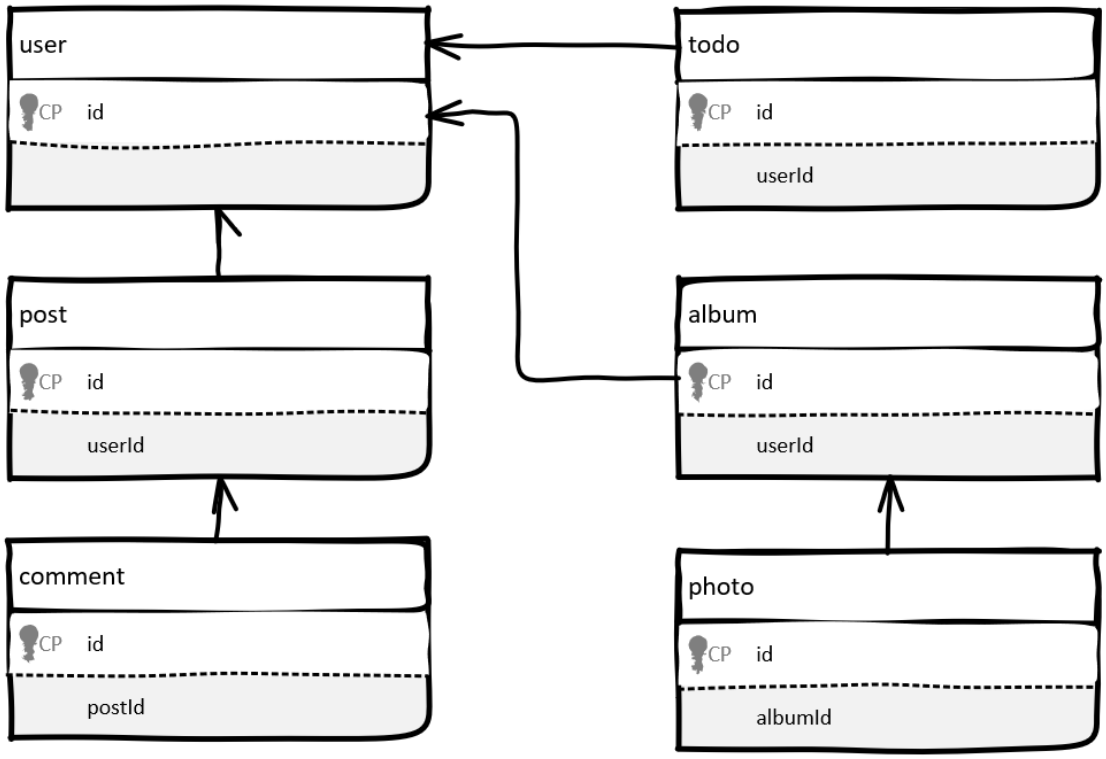
UD3E9 Consultas con el servidor

Vamos a desarrollar una aplicación basada en JavaScript que permita interactuar con la API de **jsonplaceholder.typicode.com**. El objetivo es aprender a realizar llamadas a APIs REST utilizando fetch, gestionar errores de red, persistir configuraciones en localStorage y construir una interfaz para el mantenimiento de diferentes entidades.

Modelo de datos.



Endpoints API “https://jsonplaceholder.typicode.com”

- **“/users”**. Devuelve la lista de usuarios.
- **“/todos”**. Devuelve la lista de tareas pendientes.
- **“/posts”**. Devuelve la lista de publicaciones.
- **“/comments”**. Devuelve la lista de respuestas a publicaciones.
- **“/albums”**. Devuelve la lista de álbumes.
- **“/photos”**. Devuelve la lista de fotos de los álbumes.

Adicionalmente el servicio permite las siguientes opciones de filtrado.

- Recuperar una entidad filtrando por el valor exacto del campo. “**?campo=valor**”.
- Limitar el número de datos devueltos en la petición. “**?_start=0&_limit=10**”.
    - “**_start=numero**”. Indica la posición del primer elemento a devolver.
    - “**_limit=numero**”. Indica la cantidad de elementos a devolver.

Por ejemplo.

https://jsonplaceholder.typicode.com/posts

https://jsonplaceholder.typicode.com/posts?userId=1

https://jsonplaceholder.typicode.com/posts?_start=0&_limit=10

Se pide.

Página principal (index.html)

Mostrar un listado de las entidades (user, todo, album, photo, post, comment). Cada entidad estará representada por una tarjeta con nombre de la entidad y el número total de elementos. Al hacer click sobre la tarjeta se abrirá el mantenimiento de la entidad.

Gestionar los errores al intentar cargar los datos de cada tarjeta.

NOTA: si no te has dado cuenta, tienes 6 peticiones en está página, igual hacerlas en serie no es la mejor idea.

Página de mantenimiento para cada entidad (nombre-entidad-plural.html)

Listado con paginador.

Opción para seleccionar el número de elementos por página mediante un combo (valores: 5, 10, 20, todos).

Filtro por el campo principal de cada entidad (por ej, “name” para “user”, “title” para “todo”, etc...), en este caso hay que descargar todos los valores y realizar el filtrado en el cliente.

Almacena en localStorage la preferencia de número de elementos por página para cada entidad.

Al filtrar o página se actualiza los datos sin recargar la página.

Muestra mensajes de error en caso de problemas de conexión en la propia página.

Para la entidad “user”, incluir opciones en cada entrada:

- “Ver pendientes”. Abre el mantenimiento de “todo” filtrando por el userId.
- “Ver álbumes”. Abre el mantenimiento de “album” filtrando por userId.
- “Ver posts”. Abre el mantenimiento de “post” filtrando por userId.

Para la entidad “album”, incluir en cada entrada la opción “Ver fotos” para abrir el mantenimiento de “photo” filtrando por albumId.

Para la entidad “post”, incluir en cada entrada la opción de “Ver comentarios” para abrir el mantenimiento de “comment” filtrado por postId.

Página de propiedades de cada entidad (nombre-entidad.html)

Muestra un formulario con los campos de la entidad seleccionada.

Permitir al usuario editar los datos, simulando actualizaciones mediante PUT en la API. En este caso mostrar la respuesta en un alert.

Incluye una opción “Volver” que nos devuelve al mantenimiento previo.

RETO: Si el mantenimiento previo estaba filtrado por algún “Id” ¿puedo recuperarlo?

Para la entidad “photo”, incluir:

- Miniatura que se mostrará en el formulario.
- Botón ampliar que abrirá la imagen a tamaño real en una ventana nueva (nueva no es la misma).

Arquitectura

Las peticiones al servidor debes realizarlas a través de fetch, te recuerdo que devuelve una Promise.

Una clase autoinvocada que contenga la lógica principal de la aplicación (carga de datos, configuración del paginador, manejo de errores de comunicación, etc.).

Un archivo de utilidades con funciones auxiliares para la manipulación del DOM, persistencia en localStorage, y según lo plantees sessionStorage podría ayudarte a comunicar entre ventanas.

El código JavaScript propio de cada página es independiente del código con la lógica de la aplicación. Cada página debería tener su propio código para gestionar la interfaz de usuario.

Se pueden producir errores, y puede que dependan del servidor, try-catch-finally pueden ser útiles para comunicarnos con el servidor y para gestionar la carga de información en la interfaz de usuarios. (Igual hay que forzar algún error para poder probar todos los requisitos).