1. Componente evento-item

Crea un componente llamado evento-item. Este componente representará un evento por lo que recibirá los datos del evento como atributo de entrada (**@Input**). La plantilla de este nuevo componente contendrá el interior el elemento ** ** (no incluido). La clase **card** ponla directamente al elemento **<eventoitem>.**

Además, vamos a poner un botón para borrar los eventos. Este botón se situará justo después del párrafo de la descripción, así:

```
<div class="text-end">
 <button class="btn btn-danger ml-3" (click)="deleteEvento()">Delete</button>
</div>
```

Cuando hagamos clic en el botón, el componente emitirá una señal al padre (sin valor asociado → **EventEmitter<void>**), que le indicará que debe eliminar el evento del array. No hace falta emitir ningún valor porque el componente padre puede usar la variable local del **@for** que representa al objeto del evento para saber cuál tiene que borrar.

2. Borrar un evento cuando se está filtrando¶

Por defecto en *Angular*, por razones de rendimiento, los filtros (*pipes*) se comportan como filtros puros (*pure pipes*). Esto quiere decir, que sólo actualizan los datos mostrados (ejecutan el filtro otra vez) si se cambia la referencia al objeto (o array) que se está filtrando, en lugar de si simplemente modificamos dicho objeto.

Esto sucede porque es mucho más simple (en términos de rendimiento) comprobar periódicamente si una referencia a memoria (número) ha cambiado, que analizar en profundidad dicho objeto buscando cambios. Por ello, si borramos un elemento del array con la función **splice()**, que modifica el array original, y estamos filtrando los eventos por nombres, parecerá que no se ha borrado. La vista no se actualiza porque el filtro no se vuelve a ejecutar.

Tenemos varias soluciones, pero todas implican generar un array nuevo:

- **Recomendado**: Borrar con el método filter (que devuelve un array nuevo) y reasignar el array a la variable.
- Usando splice para borrar la posición del elemento, y crear después una copia del array y reasignarla (peor solución que la anterior): this.eventos = [...this.eventos]. Para ello hemos usado el operador de propagación (*spread*).

Otra solución no recomendada (coste alto de rendimiento), es declarar el filtro como impuro, estableciendo la propiedad pure a false en el decorador @Pipe, como se puede observar en el siguiente enlace: Detecting impure changes within composite objects

3. Componente evento-add

Crea un componente llamado **evento-add** y sitúa ahí el formulario y toda la lógica del mismo (transformar la imagen a **base64**, etc.), además del HTML del formulario, claro está.

Este componente emitirá el objeto del evento al componente padre cuando enviemos el formulario. El componente padre (**eventos-show**) lo recogerá (con la variable especial $**event** en el HTML) y lo añadirá al array de eventos.

En este caso se aplica el mismo principio que al borrar. Si estamos filtrando los eventos, no actualizará los cambios (evento añadido) si no detecta un array nuevo. La solución es crear una copia del array añadiendo el nuevo elemento al final.

```
this.eventos = [...this.eventos, nuevoEvento]
```

4. Servicio evento

Crea un servicio llamado **evento** (**EventoService**), y dentro del mismo, un método **getEventos**() que devuelva los eventos iniciales, en lugar de tenerlos en el componente **eventos-list** directamente. Desde este componente accede al servicio para obtener dichos eventos.