**MOSTRAR LISTA FILTRADA POR FILTROS DE TIPO (LIST) Y NOMBRE (HEADER)**

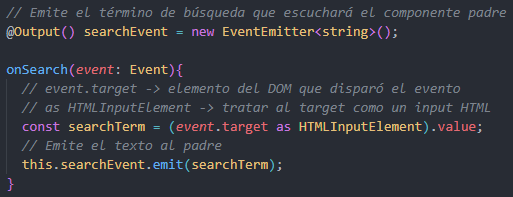
**CONEXIÓN HEADER\_COMPONENT / APP**

*El header pasa el pokemon a buscar al padre y este actualiza la propiedad del componente list (previamente filtrada por tipo) para que se busque este pokemon con la lista ya filtrada por tipo.*

1. El Header es un componente hijo
2. En el HTML de Header hay un input para buscar un pokemon
3. Este input tiene un evento (input)="onSearch($event)" para que **cada vez** que se escriba algo dentro de ese input se ejecute el método onSearch a través del ts del Header



1. Este método extrae el valor del input y lo envía al padre con el emisor @Output



1. El padre (app) escucha el evento de la siguiente manera.
2. En el componente Header del HTML del padre hay un enlace de evento (binding event) por el que recibe la información del Header.

En este caso a través del emisor searchEvent del Header, el padre ejecuta el método onSearchTermChange recibiendo el evento (searchTerm, string del pokemon a buscar) del emisor.



**CONEXIÓN APP / LIST\_COMPONENT**

*El padre va a actualizar la propiedad searchTerm (nombre del pokemon del buscador) del list\_component. También actualiza la lista de pokemon filtrados con el método filterPokemons, primero por el tipo con el método onTypeChange y después por este searchTerm.*

*Opcional: Al ir tecleando se irá poniendo el primer pokemon de la lista de pokemon filtrados en los detalles.*

1. El padre se va a coordinar con los otros componentes.
2. Utiliza un @ViewChild para obtener una referencia de PokemonListComponent

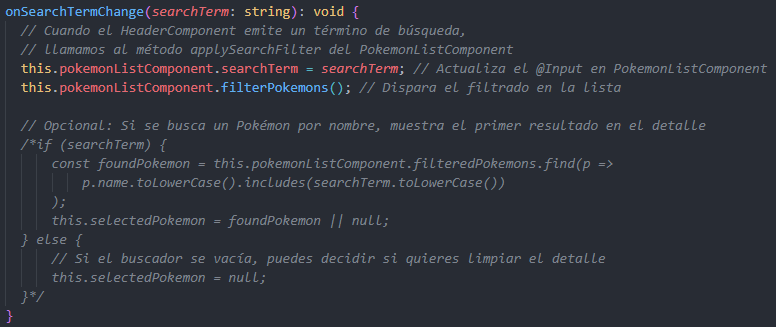


1. Con esa referencia, actualiza la propiedad searchTerm del hijo PokemonListComponent.

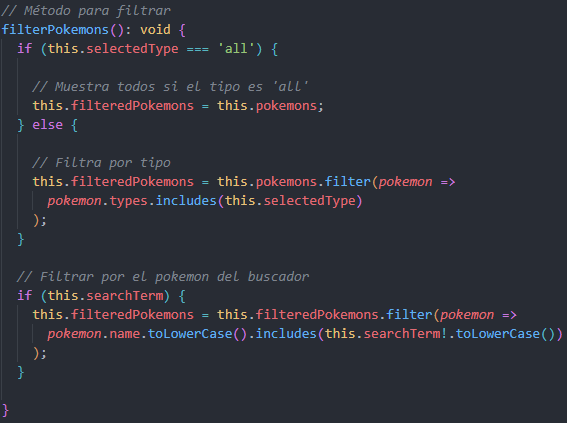
El searchTerm es un @Input que se encuentra en el ts de PokemonListComponent para recibir el pokemon a buscar desde el padre.



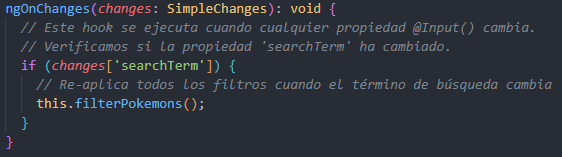
En este mismo método, aparte de actualizar la propiedad del ts de PokemonListComponent, se actualiza la lista de pokemon filtrados.



1. Como ahora ya está el searchTerm actualizado, se va a filtrar esta lista primeros por tipos y luego por el nombre (searchTerm)



1. Alternativa: Se podría añadir el siguiente método de la interfaz OnChanges para ir mirando si se va actualizando el searchTerm, y por tanto llamaría al método de filtrar. Esto se daría en el caso de que no se actualizase searchTerm desde el padre. Sería redundante. Es mejor práctica el ngOnChanges para que el list\_component sea reactivo a sus propias entradas.

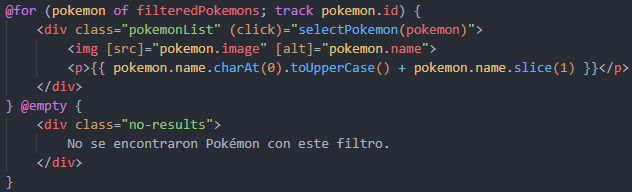


**MOSTRAR LOS DETALLES DEL POKEMON PULSANDO AL POKEMON EN LA LISTA FILTRADA**

**CONEXIÓN LIST\_COMPONENT / APP**

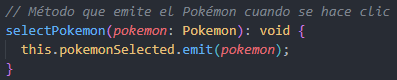
*El componente lista recoge el pokemon que ha sido pulsado con el método selectPokemon. Con un @Output se lo manda al padre.*

1. Cuando se pulsa un pokemon en la lista se ejecuta un método al que se le pasa el pokemon seleccionado.



1. Este método emitirá este pokemon al padre a través de un @Output

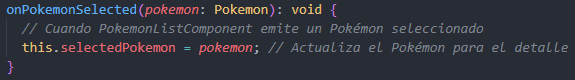




1. En el padre se guarda el pokemon en el selectedPokemon, que es el que mandará al componente de detalles.



1. Se actualiza ese selectedPokemon en el padre con del siguiente método.



**CONEXIÓN APP / DETAIL\_COMPONENT**

*El padre recibe el pokemon de la lista. Se lo manda al componente de detalles y este lo recibe con el @Input. El método ngOnChanges sirve para ir mirando si el pokemon se va actualizando y por tanto realizar alguna acción. Finalmente muestra los detalles.*

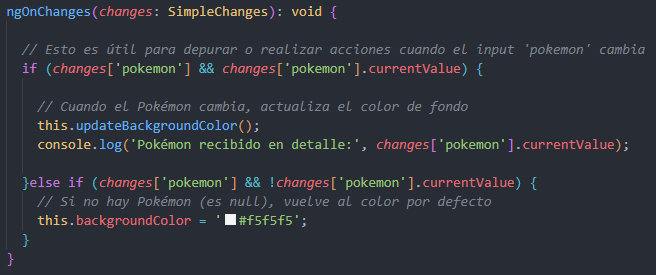
1. Se manda ese pokemon al componente detalles a través del html del padre.



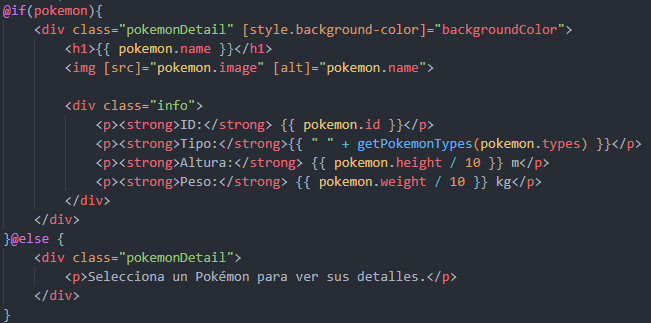
1. Se actualiza el pokemon en el componente de detalles mandado por el padre.



1. Con el siguiente método se realizan acciones cuando haya cambios en el pokemon que se envía desde el padre, en esta caso cambiar el color del fondo.



1. Muestras los detalles del pokemon en el html



MOSTRAR LA LISTA DE POKEMON FILTRADOS EN EL BUSCADOR Y AL PULSAR, SE VEAN LOS DETALLES EN EL COMPONENTE DE DETALLES