Semana 2

Curso de Angular

Ejercicio semana 2

Client-side Web Development
2nd course – DAW
IES San Vicente 2017/2018
Author: Arturo Bernal Mayordomo

Index

Listar eventos

Vamos a mostrar imágenes con los eventos. En este caso podéis descargar 2 imágenes que os proporciono en la tarea del Moodle o usar las que queráis. En cualquier caso debéis situarlas dentro de la carpeta **src/assets**, y referenciarlas en los objetos de los eventos.

```
events: IEvent[] = [{
  ...
  image: 'assets/evento1.jpg',
  ...
}, {
  ...
  image: 'assets/evento2.jpg',
  ...
}];
```

Posteriormente, hay que mostrar la imagen en la plantila HTML. Para ello situamos el elemento <img> correspondiente dentro del elemento , de la siguiente manera:

```
<div class="card">
  <img class="card-img-top">
  <div class="card-body">
    ...
```

Debéis vincular el atributo src de la imagen a la propiedad event.image del objeto correspondiente.

Además hay que filtrar el título del evento con el filtro titlecase (pone la primera letra de cada palabra en mayúscula). El precio se mostrará con el símbolo del euro mediante un filtro (ver ejemplo en apuntes) y la fecha en formato dd/MM/y (ver ejemplo en apuntes). Los eventos listados tendrán este aspecto:

Filtrar y ordenar eventos

Vamos a introducir una barra de búsqueda (igual que en el ejemplo de los apuntes) para filtrar los eventos por título. Crea un filtro (pipe) personalizado llamado **event-filter**. Recibirá el array de eventos (IEvent[]) y una cadena de búsqueda (string), y devolverá los eventos que contengan la cadena en su título (podéis incluir también la búsqueda en la descripción).

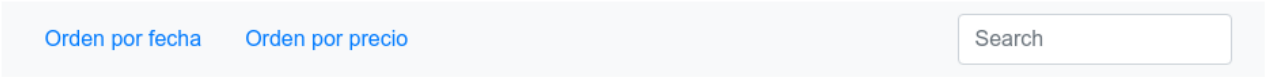
Además, vamos a crear 2 enlaces para ordenar los eventos por fecha o por precio. Al hacer clic sobre estos enlaces se llamará a un método del componente, que **borrará** el filtro actual de búsqueda y ordenará el array en base a la fecha o al precio. Para ordenar por fecha, al estar almacenada como cadena en formato yyyy-mm-dd, puedes ordenar alfabéticamente.

Este es el HTML a añadir para mostrar tanto el campo de búsqueda como los 2 enlaces:

```
<nav class="navbar navbar-light bg-light justify-content-between mt-3">
   <ul class="nav nav-pills">
     <li class="nav-item">
       <a class="nav-link" href="#" (click)="orderDate()">Orden por fecha</a>
     </li>
     <li class="nav-item">
       <a class="nav-link" href="#" (click)="orderPrice()">Orden por precio</a>
     </li>
   </ul>
   <form class="form-inline">
     <input class="form-control mr-sm-2" type="text" [(ngModel)]="search" name="search"
placeholder="Search" aria-label="Search">
   </form>
 </nav>

<div class="row">
  ... (Aquí es donde listamos los eventos)
</div>
```

¡No olvides incluir el módulo **FormsModule** en el módulo de tu aplicación!



Nuevo evento

Como último apartado del ejercicio, vamos a crear un formulario para insertar un nuevo evento. Por ahora lo crearemos en la misma página, encima del listado de eventos. Tampoco se guardarán de forma permanente. Ya iremos solucionando estos "problemas" a lo largo del curso.

En lugar de que cada campo de texto haga referencia a una variable o atributo de la clase del componente por separado. Es más recomendable crear un objeto de evento vacío y referenciar sus campos. De esta forma cuando enviemos el formulario, lo único que tendremos que hacer es añadir el objeto con sus propiedades ya rellenadas al array de eventos.

```
export class EventsShowComponent implements OnInit {

  newEvent: IEvent = {
    title: '',
    description: '',
    image: '',
    price: 0,
    date: ''
  };

  ...
}
```

Esta es la estructura del formulario que debéis implementar. Sólo he vinculado con [(ngModel)] el título del evento. Vinculad el resto vosotros/as:

```
<form class="mt-4" (ngSubmit)="addEvent()">
  <div class="form-group">
    <label for="name">Nombre</label>
    <input type="text" class="form-control" name="title" [(ngModel)]="newEvent.title"  placeholder="Enter
name">
  </div>
  <div class="form-group">
    <label for="date">Fecha</label>
    <input type="date" class="form-control" name="date">
  </div>
  <div class="form-group">
    <label for="description">Descripción</label>
    <textarea class="form-control" name="description" rows="3"></textarea>
  </div>
  <div class="form-group">
    <label for="price">Precio</label>
    <input type="number" class="form-control"  name="price" min="0.00" max="10000.00" step="0.01" />
  </div>
  <div class="form-group">
    <label for="image">Imagen</label>
    <input type="file" class="form-control" name="image" #fileImage (change)="changeImage(fileImage)">
  </div>
  <img [src]="newEvent.image" alt="" class="img-thumbnail">

  <button type="submit" class="btn btn-primary">Create</button>
</form>
```

Como se puede observar, se utiliza el evento **ngSubmit** para llamar a un método, **addEvent()** en este caso, cuando el formulario se envía. Normalmente se usaría el evento submit, pero ngSubmit tiene ciertas ventajas como la de no recargar la página automáticamente cuando se lanza. En el método **addEvent()**, debemos añadir el evento (**newEvent**) al array de eventos y reiniciar los campos del formulario otra vez. Es decir, volver a asignar a la propiedad newEvent un objeto con los campos vacíos.

No se requiere validar nada del formulario por ahora. Esa parte veremos más adelante como gestionarla con Angular.

Mostrar la descripción del evento

Al añadir un evento directamente desde el formulario, los saltos de línea estarán representados por el carácter **n** en lugar de la etiqueta **
**. Para que se reconozcan estos saltos de línea en HTML, podemos hacer 2 cosas.

Mediante la propiedad de CSS **white-space**:

```
.card-text {
  white-space: pre-wrap;
}
```

Usando el atributo **innerText** en lugar de la interpolación:

<p class="card-text"**>**{{event.description}}**</p>** → **Sólo reconoce el
**

<p class="card-text" [innerText]="event.description"**></p>** → **Reconoce el \n como salto**

Convertir la imagen a Base64 y previsualizarla

Desde un formulario, el navegador no deja acceder a la ruta local del archivo cargado (imagen), por lo que la única opción que nos queda es transformar esta imagen a Base64 (string).

Si observas el formulario, verás que el input de la imagen tiene un atributo llamado #fileImage (ya entenderemos que significan más adelante), y el evento (**change**), para cuando cambia el archivo. Este es el método implementado:

```
changeImage(fileInput: HTMLInputElement) {
    if (!fileInput.files || fileInput.files.length === 0) { return; }
    const reader: FileReader = new FileReader();
    reader.readAsDataURL(fileInput.files[0]);
    reader.addEventListener('loadend', e => {
      this.newEvent.image = reader.result;
    });
}
```

La imagen convertida la guardará automáticamente en **this.newEvent.image**. Esta imagen en Base64 se puede visualizar directamente en el campo **src** de una imagen como se puede observar en el elemento <img> del formulario.