

# Documentación Spotify

## *Funcionalidad botón like*

Añadimos funcionalidad al botón like situado al lado de la canción que esté en reproducción.

### 1. Creación del FavouriteService

- Creamos un servicio que maneja el estado de las canciones favoritas.
- Usa BehaviorSubject para mantener una lista reactiva de canciones favoritas.
- Almacena los favoritos en localStorage para persistencia.

```
1 import { Injectable } from '@angular/core';
2 import { BehaviorSubject } from 'rxjs';
3 import { TrackModel } from '@core/models/tracks.model';
4
5 @Injectable({
6   providedIn: 'root'
7 })
8 export class FavoriteService {
9   private favoriteTracksSubject = new BehaviorSubject<TrackModel[]>([]);
10  public favoriteTracks$ = this.favoriteTracksSubject.asObservable();
11
12  constructor() {
13    this.loadFavoritesFromStorage();
14  }
15
16  toggleFavorite(track: TrackModel): void {
17    const currentFavorites = this.favoriteTracksSubject.value;
18    const trackIndex = currentFavorites.findIndex(t => t._id === track._id);
19
20    if (trackIndex === -1) {
21      currentFavorites.push(track);
22    } else {
23      currentFavorites.splice(trackIndex, 1);
24    }
25
26    this.favoriteTracksSubject.next(currentFavorites);
27    localStorage.setItem('favoriteTracks', JSON.stringify(currentFavorites));
28  }
29
30  isFavorite(trackId: string): boolean {
31    return this.favoriteTracksSubject.value.some(track => track._id === trackId);
32  }
33
34  private loadFavoritesFromStorage(): void {
35    const stored = localStorage.getItem('favoriteTracks');
36    if (stored) {
37      this.favoriteTracksSubject.next(JSON.parse(stored));
38    }
39  }
40 }
```

## 2. Componente MediaPlayer

- Agregamos el botón de like en el reproductor.
- Implementamos dos métodos principales:

```
1 toggleFavorite(): void {
2     const currentTrack = this.multimediaService.trackInfo$.getValue();
3     if (currentTrack) {
4         this.favoriteService.toggleFavorite(currentTrack);
5     }
6 }
7
8 isFavorite(): boolean {
9     const currentTrack = this.multimediaService.trackInfo$.getValue();
10    return currentTrack
11        ? this.favoriteService.isFavorite(currentTrack._id)
12        : false;
13 }
```

## 3. FavoritePage Component

- Creamos el componente que muestra las canciones favoritas.
- Nos suscribimos al observable de favoritos:

```
1 export class FavoritePageComponent {
2     tracks$: Observable<TrackModel[]>;
3
4     constructor(private favoriteService: FavoriteService) {
5         this.tracks$ = this.favoriteService.favoriteTracks$;
6     }
7 }
```

## 4. Template de FavoritePage

- Usamos el async pipe para mostrar las canciones.
- El operador ?? proporciona un array vacío como fallback:

```
1 <div class="favorites-page">
2   <app-play-list-header></app-play-list-header>
3   <app-play-list-body [tracks]="(tracks$ | async) ?? []"></app-play-list-body>
4 </div>
```

## Funcionalidad buscador de canciones

Añadimos funcionalidad a la barra de búsqueda para buscar canciones por nombre.

### 1. Estructura de componentes

Tenemos dos componentes principales:

- SearchComponent : Maneja el input de búsqueda.
- HistoryPageComponent : Gestiona la lista de canciones y la lógica de filtrado.

```
1 import { Component, EventEmitter, Output } from '@angular/core';
2
3 @Component({
4   selector: 'app-search',
5   templateUrl: './search.component.html',
6   styleUrls: ['./search.component.css']
7 })
8 export class SearchComponent {
9   @Output() searchEmitter: EventEmitter<string> = new EventEmitter<string>();
10   src: string = '';
11
12   callSearch(term: string): void {
13     if (term.length >= 1) {
14       this.searchEmitter.emit(term);
15     }
16   }
17 }
```

```
1 import { Component, OnInit } from '@angular/core';
2 import { TrackModel } from '@core/models/tracks.model';
3 import { SearchService } from '@modules/history/services/search.service';
4
5 @Component({
6   selector: 'app-history-page',
7   templateUrl: './history-page.component.html'
8 })
9 export class HistoryPageComponent implements OnInit {
10   tracks: Array<TrackModel> = [];
11   filteredTracks: Array<TrackModel> = [];
12
13   constructor(private searchService: SearchService) {}
14
15   ngOnInit(): void {
16     this.searchService.getAllTracks().subscribe((response: any) => {
17       const { data } = response;
18       this.tracks = data;
19       this.filteredTracks = data;
20     });
21   }
22
23   searchTracks(term: string): void {
24     if (!term) {
25       this.filteredTracks = this.tracks;
26       return;
27     }
28
29     this.filteredTracks = this.tracks.filter(track =>
30       track.name.toLowerCase().includes(term.toLowerCase())
31     );
32   }
33 }
```

## 2. Servicio de búsqueda

```
1 import { HttpClient } from '@angular/common/http';
2 import { Injectable } from '@angular/core';
3 import { Observable } from 'rxjs';
4 import { TrackModel } from '@core/models/tracks.model';
5 import { environment } from 'src/environments/environment';
6
7 @Injectable({
8   providedIn: 'root'
9 })
10 export class SearchService {
11   private readonly URL = environment.api;
12
13   constructor(private http: HttpClient) { }
14
15   getAllTracks$(): Observable<TrackModel[]> {
16     return this.http.get<TrackModel[]>(`${this.URL}/tracks`);
17   }
18 }
```

## 3. Flujo de trabajo

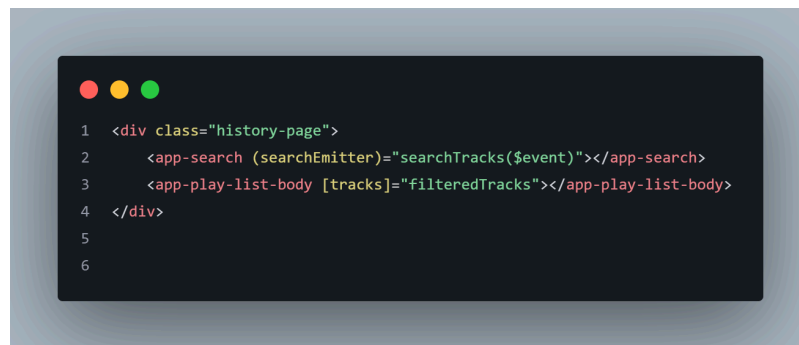
Cuando el usuario escribe en el buscador:

- El SearchComponent emite el término de búsqueda.
- El HistoryPageComponent recibe el término.
- Se filtran las canciones que coinciden con el término.
- La lista filtrada se muestra automáticamente.

## 4. Funcionalidad de filtrado

```
1 searchTracks(term: string): void {
2   if (!term) {
3     this.filteredTracks = this.tracks;
4     return;
5   }
6
7   this.filteredTracks = this.tracks.filter(track =>
8     track.name.toLowerCase().includes(term.toLowerCase())
9   );
10 }
```

## 5. Conexión con el template



## 6. Características importantes

- Búsqueda en tiempo real.
- Case-insensitive (no distingue mayúsculas/minúsculas).
- Mantiene la lista original mientras muestra resultados filtrados.
- Restaura la lista completa cuando se borra el término de búsqueda.

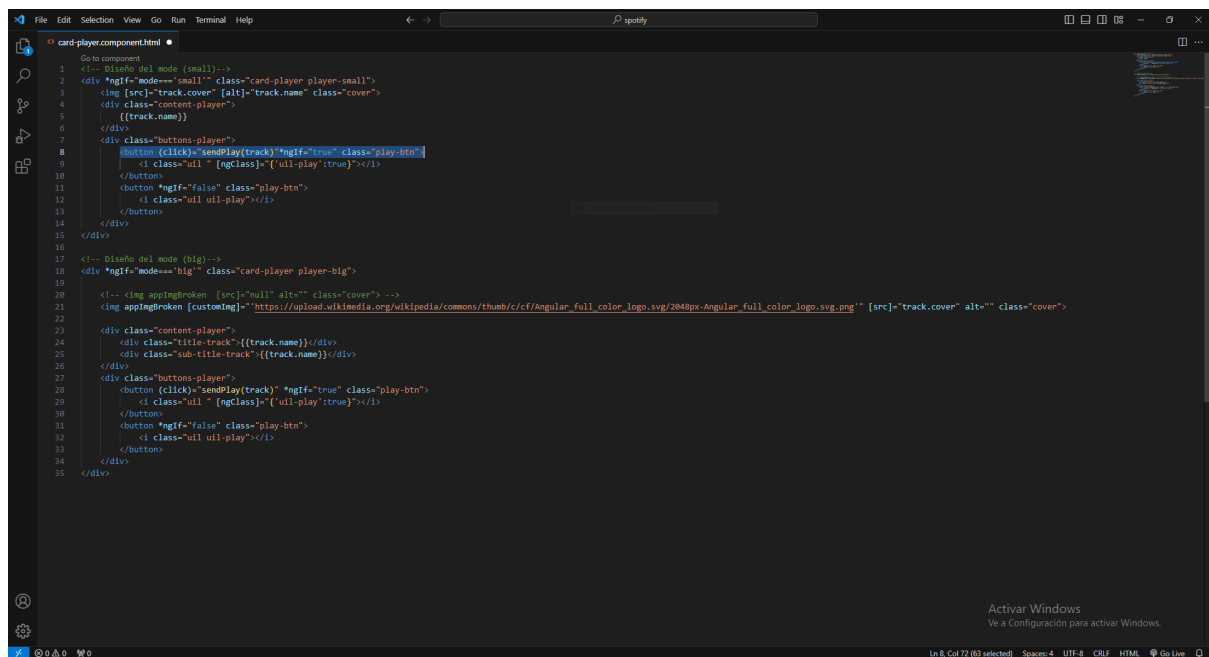
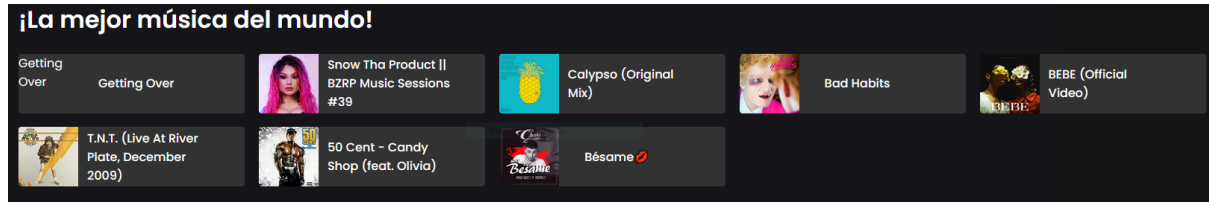
## 7. Manejo de datos

- Se cargan todas las canciones al inicio.
- Se mantienen dos arrays:
  - tracks : Lista completa original
  - filteredTracks : Lista filtrada que se muestra

Esta implementación permite una búsqueda eficiente y una experiencia de usuario fluida al buscar canciones por nombre.

## Funcionalidad a canciones de arriba

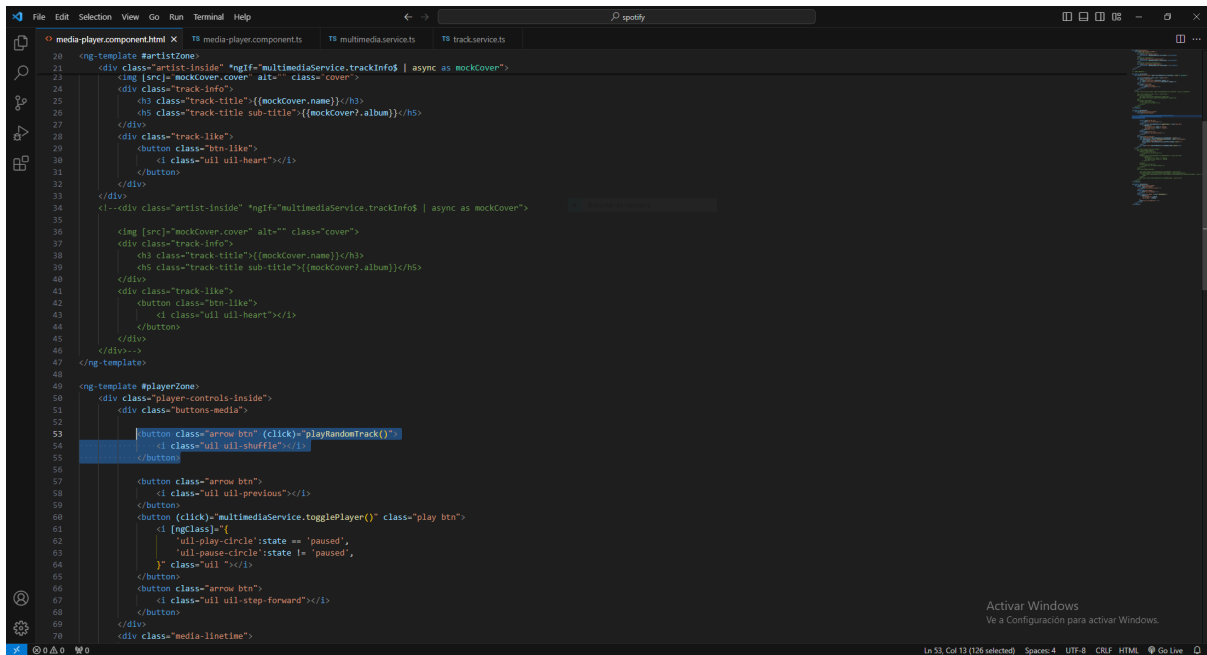
Añadimos funcionalidad a las canciones del player list small para que al pulsar en el botón de play también empiecen a reproducirse.



## Funcionalidad botón aleatorio

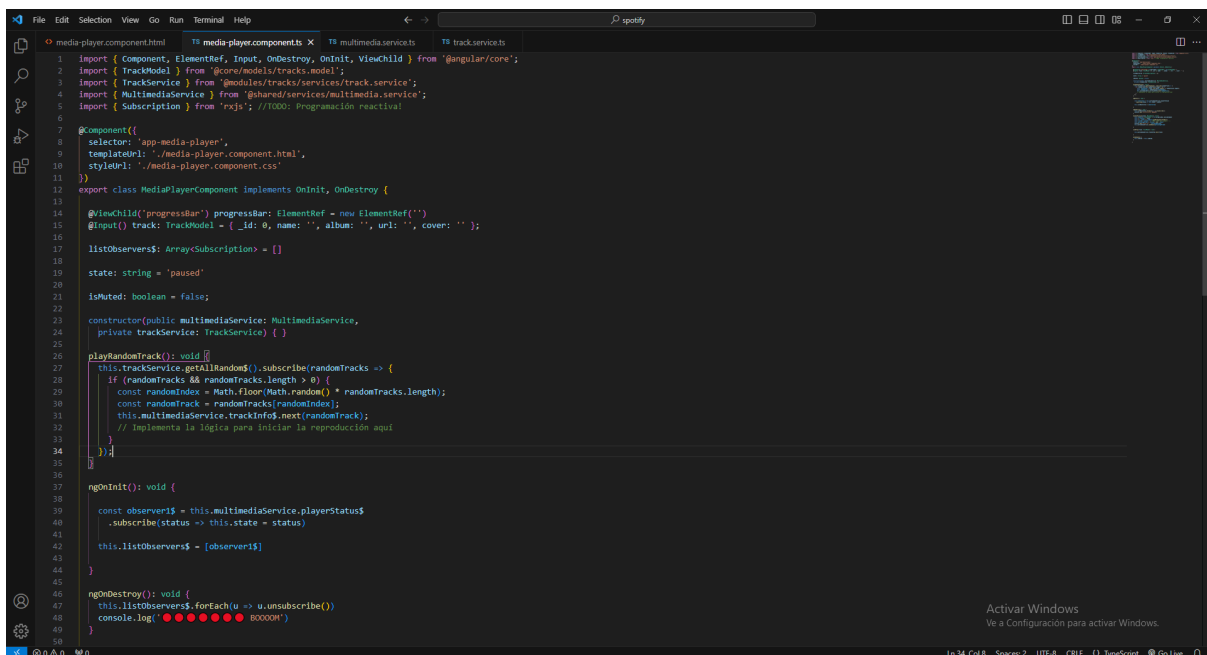
Creamos un botón aleatorio para que al pulsar en él se empiece a reproducir una canción aleatoria





```
20 <ng-template #artistZone>
21   <div class="artist-inside" *ngIf="multimediaService.trackInfo$ | async as mockCover">
22     <img [src]="mockCover.cover" alt="" class="cover">
23     <div class="track-info">
24       <h3 class="track-title">{{mockCover.name}}</h3>
25       <h5 class="track-title sub-title">{{mockCover.album}}</h5>
26     </div>
27     <div class="track-like">
28       <button class="btn-like">
29         <i class="uil uil-heart"></i>
30       </button>
31     </div>
32   </div>
33 </ng-template>
34 <!--div class="artist-inside" *ngIf="multimediaService.trackInfo$ | async as mockCover">
35   <img [src]="mockCover.cover" alt="" class="cover">
36   <div class="track-info">
37     <h3 class="track-title">{{mockCover.name}}</h3>
38     <h5 class="track-title sub-title">{{mockCover.album}}</h5>
39   </div>
40   <div class="track-like">
41     <button class="btn-like">
42       <i class="uil uil-heart"></i>
43     </button>
44   </div>
45 </div-->
46 </ng-template>
47
48 <ng-template #playerZone>
49   <div class="player-controls-inside">
50     <div class="buttons-media">
51       <button class="arrow btn" (click)="playRandomTrack()"
52         <i class="uil uil-shuffle"></i>
53       </button>
54       <button class="arrow btn">
55         <i class="uil uil-previous"></i>
56       </button>
57       <button (click)="multimediaService.togglePlayer()" class="play btn">
58         <ngclass>{{
59           'uil-play-circle':state == 'paused',
60           'uil-pause-circle':state != 'paused',
61         }} class="uil"></i>
62       </button>
63       <button class="arrow btn">
64         <i class="uil uil-step-forward"></i>
65       </button>
66     </div>
67   </div>
68   <div class="media-lintime">
```

Le añadimos la función al botón de que cuando se haga click ejecute la función playRandomTrack()



```
1 import { Component, ElementRef, Input, OnDestroy, OnInit, ViewChild } from '@angular/core';
2 import { TrackModel } from '@core/models/tracks.model';
3 import { TrackService } from '@modules/tracks/services/track.service';
4 import { MultimediaService } from '@shared/services/multimedia.service';
5 import { Subscription } from 'rxjs'; //TODOS: Programación reactiva
6
7 @Component({
8   selector: 'app-media-player',
9   templateUrl: './media-player.component.html',
10   styleUrls: ['./media-player.component.css']
11 })
12 export class MediaPlayerComponent implements OnInit, OnDestroy {
13
14   @ViewChild('progressBar') progressBar: ElementRef = new ElementRef('');
15   @Input() track: TrackModel = { _id: 0, name: '', album: '', url: '', cover: '' };
16
17   listObservers$: Array<Subscription> = [];
18
19   state: string = 'paused';
20
21   isMuted: boolean = false;
22
23   constructor(public multimediaService: MultimediaService,
24     private trackService: TrackService) {}
25
26   playRandomTrack(): void {
27     this.trackService.getAllRandoms().subscribe(randomTracks => {
28       if (randomTracks && randomTracks.length > 0) {
29         const randomIndex = Math.floor(Math.random() * randomTracks.length);
30         const randomTrack = randomTracks[randomIndex];
31         this.multimediaService.trackInfo$.next(randomTrack);
32         // Implementa la lógica para iniciar la reproducción aquí
33       }
34     });
35   }
36
37   ngOnInit(): void {
38     const observer$ = this.multimediaService.playerStatus$
39       .subscribe(status => this.state = status);
40     this.listObservers$ = [...this.listObservers$, observer$];
41   }
42
43   ngOnDestroy(): void {
44     this.listObservers$.forEach(u => u.unsubscribe());
45     console.log('●●●●●●●● BOOOOM!');
46   }
47 }
```

Esta función la hemos creado en el media-player.component.ts haciendo un input de TrackModel con el formato de las canciones e importando TrackService que es donde se encuentran. Posteriormente en el propio método playRandomTrack() creamos la estructura para reproducir una canción aleatoria.

## Funcionalidad botón repeat

**Creamos un botón repeat para que al pulsar en él se vuelva a reproducir desde el principio la canción que se está reproduciendo**



```

File Edit Selection View Go Run Terminal Help
TS track.service.ts TS media-player.component.ts # media-player.component.css media-player.component.html X
20 <ng-template #artistZone>
41 <div class="track-like">
42 <button class="btn-like">
43 <i class="uil uil-heart"></i>
44 </button>
45 </div>
46 </div><-->
47 </ng-template>
48
49 <ng-template #playerZone>
50 <div class="player-controls-inside">
51 <div class="buttons-media">
52
53 <button class="arrow btn" (click)="playRandomTrack()">
54 <i class="uil uil-shuffle"></i>
55 </button>
56
57 <button class="arrow btn">
58 <i class="uil uil-previous"></i>
59 </button>
60 <button (click)="multimediaService.togglePlayer()" class="play btn">
61 <i ngClass="{
62 'uil-play-circle': state == 'paused',
63 'uil-pause-circle': state != 'paused',
64 'class': 'uil' }"></i>
65 </button>
66 <button class="arrow btn">
67 <i class="uil uil-step-forward"></i>
68 </button>
69
70 <button class="btn-repeat" (click)="restartTrack()">
71 <i class="uil uil-repeat"></i>
72 </button>
73
74 </div>
75 <div class="media-linetime">
76 <div class="time">{{(multimediaService.timeElapsed$ | async)}}</div>
77 <span #progressBar (click)="handlePosition($event)" class="time-progress">
78 <span class="time-progress-live"
79 [style.width]="(multimediaService.playerPercentage$ | async)+"%">
80 </span>
81 </span>
82 <div class="time">{{(multimediaService.timeRemainings$ | async)}}</div>
83 </div>
84 </div>
85 <!--<div class="player-controls-inside">
86 <div class="buttons-media">
87 <button class="arrow btn">
88 <i class="uil uil-previous"></i>

```

**Le añadimos la función al botón de que cuando se haga click ejecute la función restartTrack()**

[illegible]



Accedemos al objeto audio en el MultimediaService. Al establecer currentTime a 0, la canción se reinicia desde el principio. Luego, el método play asegura que la canción se reproduzca inmediatamente después de ser reiniciada.

## Funcionalidad botón mute

Añadimos al botón de volumen que cuando se pulse cambie de forma a sin ondas o con ondas y tenga la funcionalidad de mutear o no la canción.



```
File Edit Selection View Go Run Terminal Help
spotify
TS track.service.ts TS media-player.component.ts TS multimedia.service.ts # media-player.component.css media-player.component.html
49 <ng-template #playerZone>
104 <span #progressBar (click)="handlePosition($event)" class="time-progress">
105 <span class="time-progress-live" [style.width]="(multimediaService.playerPercentage$ | async)%">
106 </span>
107 </span>
108 <div class="time">{{(multimediaService.timeRemaining$ | async)}}</div>
109 </div>
110 </ng-template>
111
112
113 <ng-template #playerAudio>
114 <div class="player-audio-inside">
115 <button class="btn-media">
116 <i class="uil uil-list-ul-alt"></i>
117 </button>
118 <button class="btn-media">
119 <i class="uil uil-boombox"></i>
120 </button>
121
122 <button class="btn-media" (click)="muteUnmute()">
123 <i [ngClass]="{
124 'uil-volume-off': isMuted,
125 'uil-volume': !isMuted,
126 }" class="uil"></i>
127 </button>
128
129 </div>
130 </ng-template>
131
```

Le añadimos la función al botón de que cuando se haga click ejecute la función muteUnmute()

```
File Edit Selection View Go Run Terminal Help
spotly

TS track.services TS media-player.component.ts TS multimedia.service.ts # media-player.component.css media-player.component.html

9 export class MultimediaService {
10   private setRemaining(currentTime: number, duration: number): void {
11     let timeLeft = duration - currentTime
12
13     let seconds = Math.floor(timeLeft % 60) //TODD: 1, 2, 5, 18... números enteros. Obtenemos segundos
14     let minutes = Math.floor((timeLeft / 60) % 60) //TODD: Obtenemos los minutos en números enteros
15
16     //TODD: 00:00 ----> 01:05 ----> 10:15
17     const displaySeconds = (seconds < 10) ? `0${seconds}` : seconds
18     const displayMinutes = (minutes < 10) ? `0${minutes}` : minutes
19
20     const displayFormat = `${displayMinutes}:${displaySeconds}`
21     this.timeRemaining$.next(displayFormat)
22   }
23
24   //TODD: Funciones publicas
25   public setAudio(track: TrackModel): void {
26     console.log('🎵 🎵 🎵 🎵', track)
27     this.audio.src = track.url
28     this.audio.play()
29   }
30
31   public togglePlayer(): void {
32     (this.audio.paused) ? this.audio.play() : this.audio.pause()
33   }
34
35   public seekAudio(percentage: number): void {
36     const ( duration ) = this.audio
37     const percentageToSecond = (percentage * duration) / 100
38     this.audio.currentTime = percentageToSecond
39   }
40
41   toggleMute(): void {
42     this.audio.muted = !this.audio.muted;
43   }
44 }
```

**Añadimos el método para Mute/Unmute en el multimedia.service.ts**

```
File Edit Selection View Go Run Terminal Help
spotly

TS track.services TS media-player.component.ts TS multimedia.service.ts # media-player.component.css media-player.component.html

12 export class MediaPlayerComponent implements OnInit, OnDestroy {
13   restartTrack(): void {
14     this.multimediaService.audio.currentTime = 0;
15     this.multimediaService.audio.play();
16   }
17
18   ngOnInit(): void {
19     const observer1$ = this.multimediaService.playerStatus$
20       .subscribe(status => this.state = status)
21     this.listObservers$ = [observer1$]
22   }
23
24   ngOnDestroy(): void {
25     this.listObservers$.forEach(u => u.unsubscribe())
26     console.log('🔴🔴🔴🔴🔴🔴🔴🔴 BOOOOM!')
27   }
28
29   handlePosition(event: MouseEvent): void {
30     const elNative: HTMLElement = this.progressBar.nativeElement
31     const ( client ) = event
32     const ( x, width ) = elNative.getBoundingClientRect()
33     const clickX = clientX - x //TODD: 1500 (por ejemplo) - x
34     const percentageFromX = (clickX + 100) / width
35     console.log('Click (x):', percentageFromX)
36     this.multimediaService.seekAudio(percentageFromX)
37   }
38
39   muteUnmute(): void {
40     this.isMuted = !this.isMuted;
41     this.multimediaService.toggleMute();
42   }
43 }
```

Añadimos el método `muteUnmute()` en `media-player.component.ts` para que actualice el estado `isMuted` y llame al método `toggleMute` del `multimedia.service.ts`

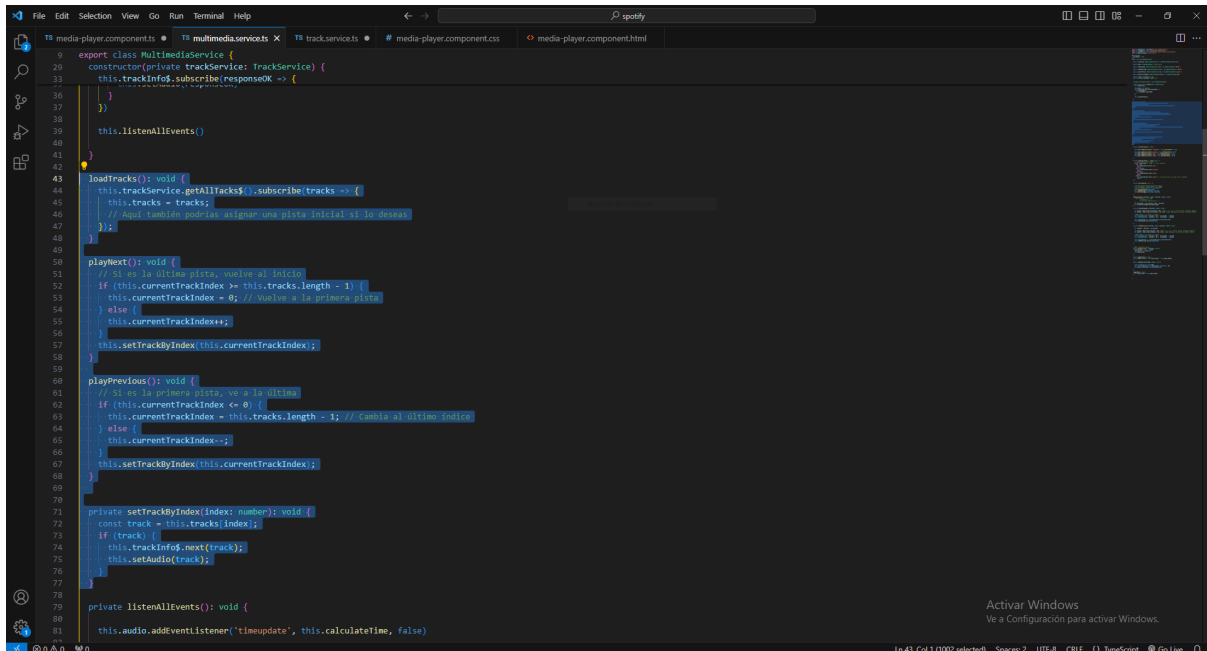
## *Funcionalidad botón flechas anterior y siguiente*

```
export class MultimediaService {  
  public trackInfo$: BehaviorSubject<any> = new BehaviorSubject(undefined)  
  public audio!: HTMLAudioElement //TODO:<audio>  
  public timeElapsed$: BehaviorSubject<string> = new BehaviorSubject('00:00')  
  public timeRemaining$: BehaviorSubject<string> = new BehaviorSubject('-00:00')  
  public playerStatus$: BehaviorSubject<string> = new BehaviorSubject('paused')  
  public playerPercentage$: BehaviorSubject<number> = new BehaviorSubject(0)  
  public tracks: TrackModel[] = [];  
  private currentTrackIndex: number = 0;  
}
```

Añadimos las propiedades `track` y `currentTrackIndex` en la clase `MultimediaService`

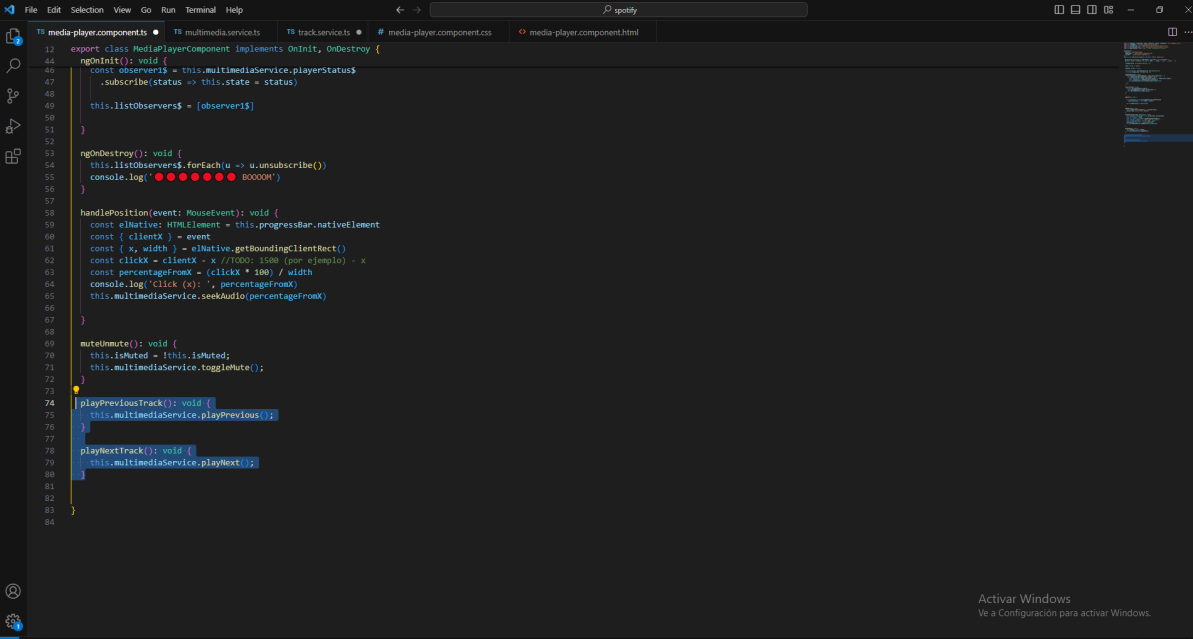
```
constructor(private trackService: TrackService) {  
  this.loadTracks();  
  
  this.audio = new Audio()  
  this.trackInfo$.subscribe(responseOK => {  
    if (responseOK) {  
      this.setAudio(responseOK)  
    }  
  })  
  
  this.listenAllEvents()  
}
```

**Agregamos el método loadTracks() al constructor.**



```
9 export class MultimediaService {
20   constructor(private trackService: TrackService) {
33     this.trackInfo$.subscribe(responseOK => {
36     })
37   }
38   this.listenAllEvents()
40 }
41
42
43 loadTracks(): void {
44   this.trackService.getAllTracks().subscribe(tracks => {
45     this.tracks = tracks;
46     // Aquí vamos a poder asignar una pista inicial si lo desea
47   });
48 }
49
50 playNext(): void {
51   // Si es la última pista, vamos al inicio
52   if (this.currentTrackIndex >= this.tracks.length - 1) {
53     this.currentTrackIndex = 0; // Vamos a la primera pista
54   } else {
55     this.currentTrackIndex++;
56   }
57   this.setTrackByIndex(this.currentTrackIndex);
58 }
59
60 playPrevious(): void {
61   // Si es la primera pista, va a la última
62   if (this.currentTrackIndex <= 0) {
63     this.currentTrackIndex = this.tracks.length - 1; // Cambia al último índice
64   } else {
65     this.currentTrackIndex--;
66   }
67   this.setTrackByIndex(this.currentTrackIndex);
68 }
69
70 private setTrackByIndex(index: number): void {
71   const track = this.tracks[index];
72   if (track) {
73     this.trackInfo$.next(track);
74     this.setAudio(track);
75   }
76 }
77
78 private listenAllEvents(): void {
79   this.audio.addEventListener('timeupdate', this.calculateTime, false)
80 }
81 }
```

**Incluimos los métodos que van a manejar la lógica de las flechas en el MultimediaService.**



```
12 export class MediaPlayerComponent implements OnInit, OnDestroy {
13
14   ngOnInit(): void {
15     const observer1 = this.multimediaService.playerStatus$
16       .subscribe(status => this.state = status)
17
18     this.listObservers$ = [observer1]
19   }
20
21   ngOnDestroy(): void {
22     this.listObservers$.forEach(u => u.unsubscribe())
23     console.log('BOOOOM!')
24   }
25
26   handlePosition(event: MouseEvent): void {
27     const elNative: HTMLElement = this.progressBar.nativeElement
28     const { clientX } = event
29     const { x, width } = elNative.getBoundingClientRect()
30     const clickX = clientX - x // TODO: 1500 (por ejemplo) - x
31     const percentageFromX = (clickX * 100) / width
32     console.log('Click (x): ', percentageFromX)
33     this.multimediaService.seekAudio(percentageFromX)
34   }
35
36   muteMute(): void {
37     this.isMuted = !this.isMuted
38     this.multimediaService.toggleMute()
39   }
40
41   playPreviousTrack(): void {
42     this.multimediaService.playPrevious()
43   }
44
45   playNextTrack(): void {
46     this.multimediaService.playNext()
47   }
48 }
```

Por último agregamos los métodos al MediaPlayerComponent que van a llamar a los métodos del MultimediaService.