**Activity 3**

Noah Funderburgh

Grand Canyon University

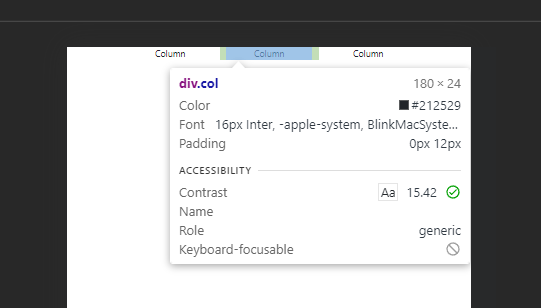
Course Number: CST-323-O500

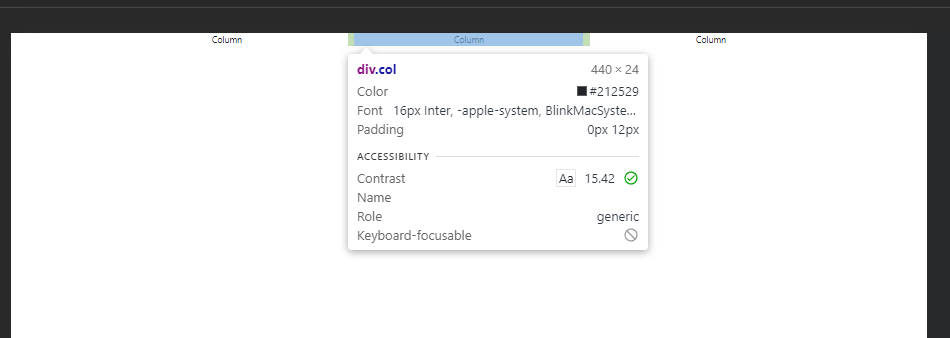
Jason Jazzar

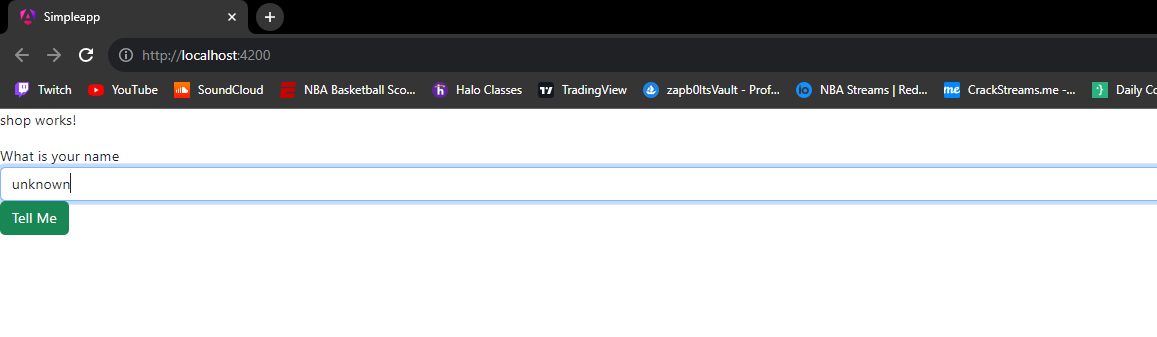
11/26/2023

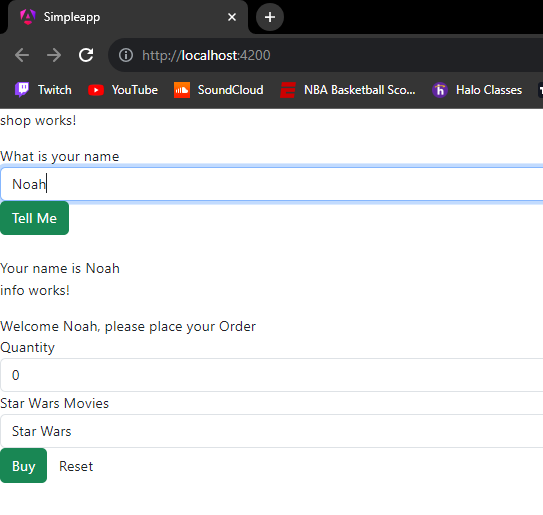
**Part 1**

**Bootstrap Small Screen**

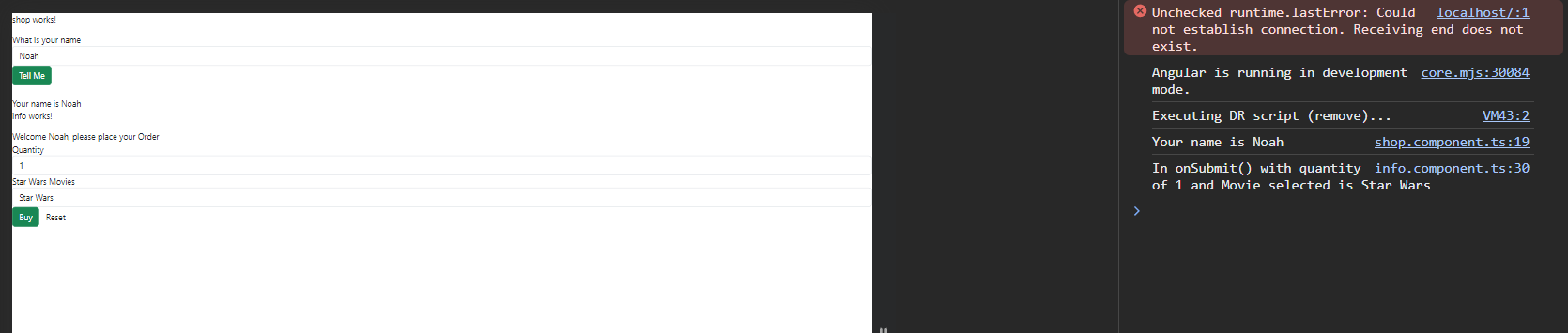
****

**Bootstrap Large Screen**

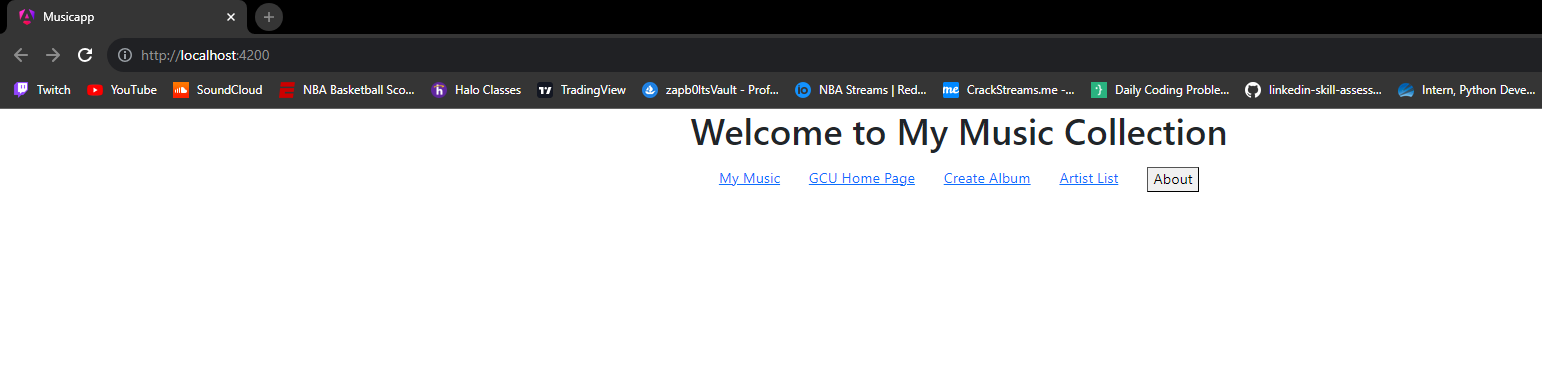
**Before Name Entered**

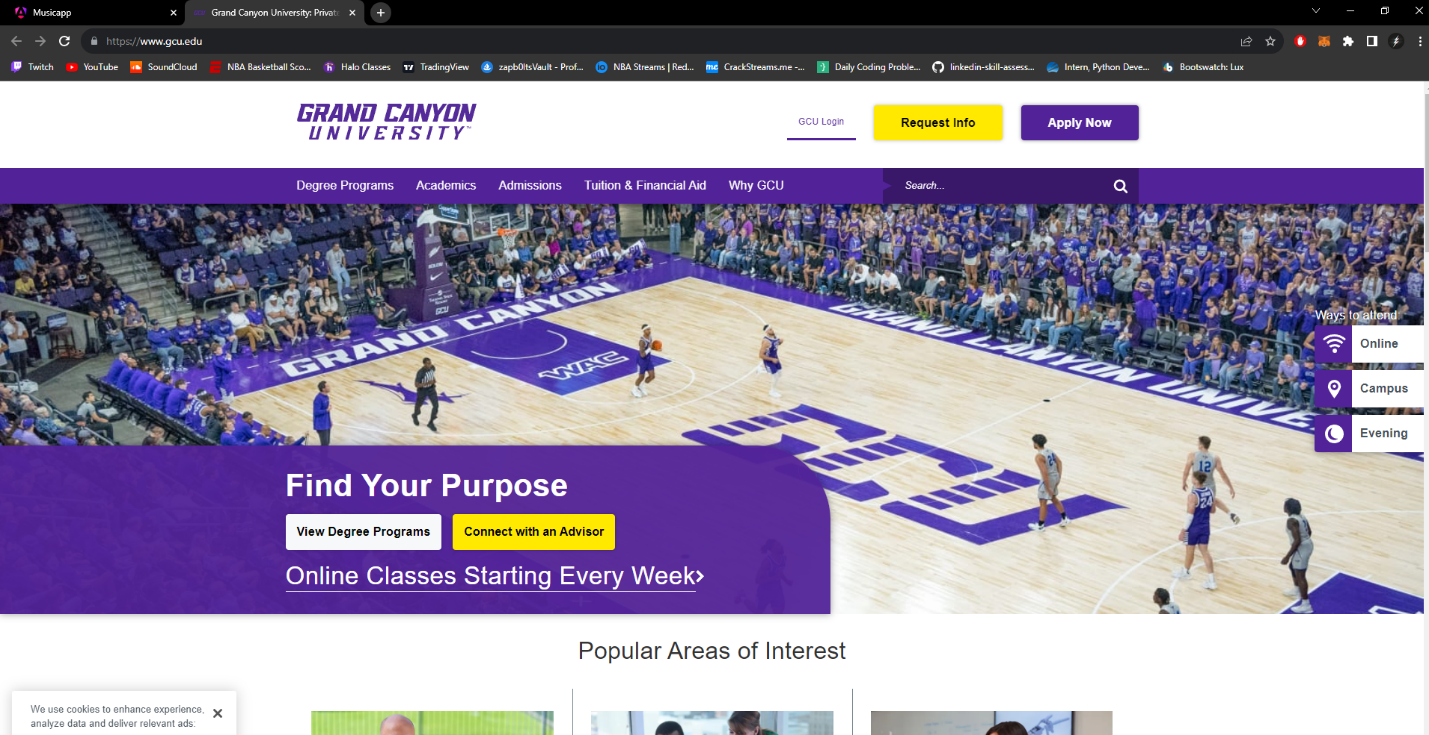
**After Name Entered**

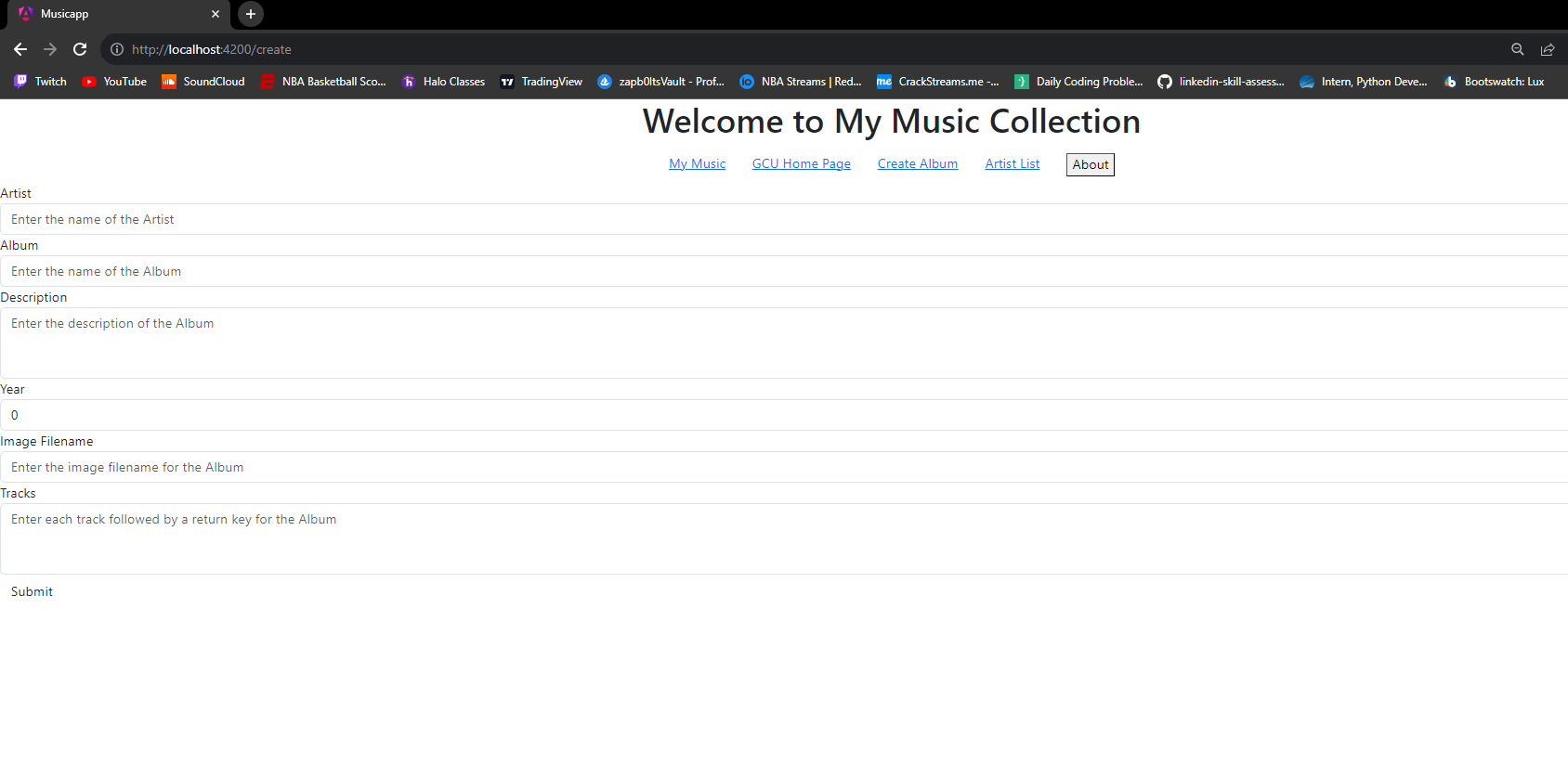
**Dev tools with info in console**

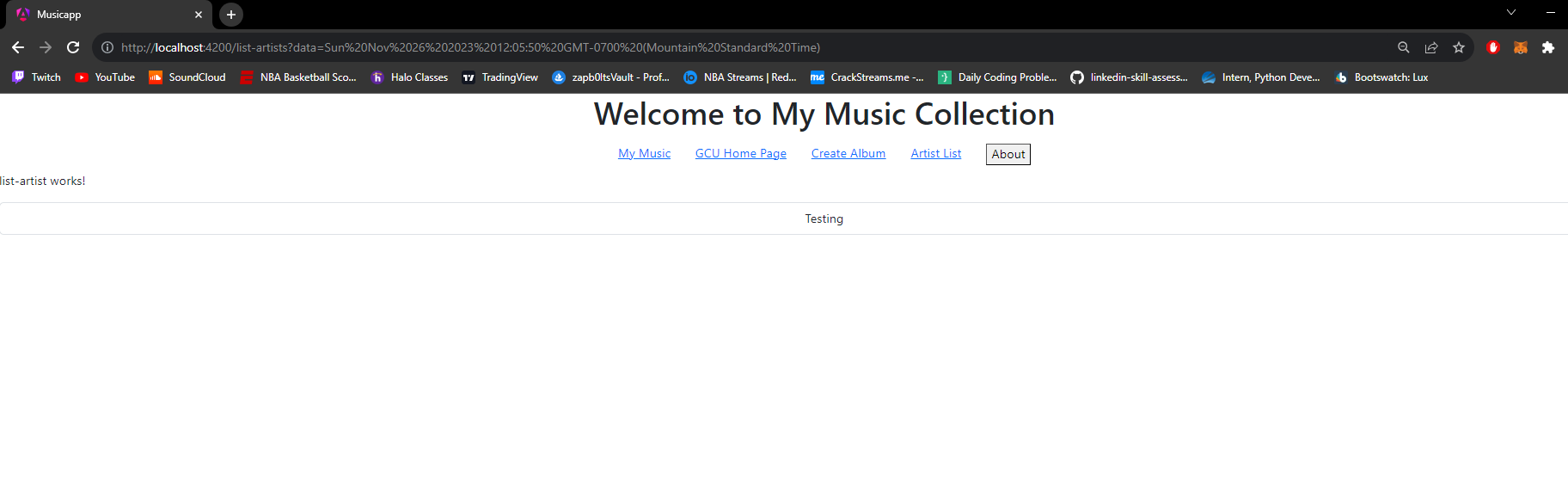
****

**Part 2**

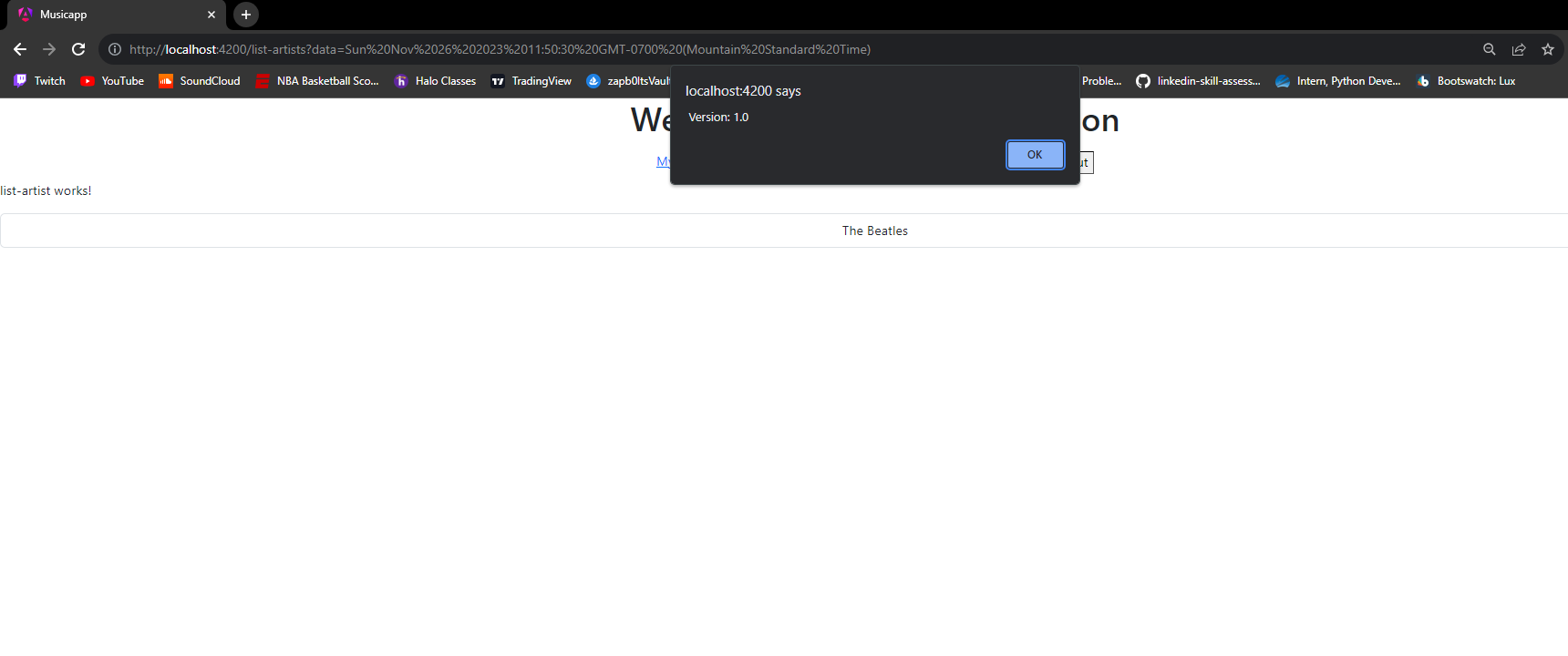
**Initial Page**

**GCU Home Page**

**Create Album page**

**Artist List page**

**About Box**

****

**Research Questions**

1. Describe @Input decorator used in info.component.ts

The @Input decorator is for passing data from the parent to the child component. By doing so we can data bind the value of the parentData in the parent component to data. This allows us to use the data for its functionality and display.

1. Describe [value] used in info.component.html

[value] is a way of property binding the value of an attribute or property to an HTML element. This moves a value in one direction from a component property into the targeted element property.

1. Describe [(ngModel)] also used in info.component.html

[(ngModel)] creates a Form control that can be used as a two way bind that creates a bidirectional flow of data between both typescript and html. The form control tracks the value, user interaction and validation status of the control and keeps the view synced with the model.

1. Add complete comments to music-service.service.ts. Include this commented code in a Microsoft Word document or a text document. Make sure your submission reads like a code file.

import { Injectable } from '@angular/core';

import exampledata from '../../data/sample-music-data.json';

import { Album } from '../models/Album';

import { Artist } from '../models/Artist';

import { Track } from '../models/Track';

/\* The `@Injectable()` decorator with the `providedIn: 'root'` option is used to provide the service at the root level of

the application. This means that the service will be available as a singleton instance throughout the entire

application. It allows the service to be injected into any component or service without the need to manually add it to

the providers array in the module. \*/

@Injectable({

providedIn: 'root'

})

/\* The MusicServiceService class manages artists and albums, allowing users to retrieve, create, update, and delete albums. \*/

export class MusicServiceService {

private readonly artists: Artist[] = [];

private readonly albums: Album[] = [];

/\*\*

\* The constructor function initializes the class by calling the createArtists and createAlbums methods.

\*/

constructor() {

this.createArtists();

this.createAlbums();

}

/\*\*

\* The function creates a new artist object and adds it to the artists array.

\*/

private createArtists(): void {

this.artists.push(new Artist(0, 'The Beatles'));

}

/\*\*

\* The function creates albums by filtering data for tracks by "The Beatles" and then creating album objects with the

\* filtered tracks.

\*/

private createAlbums(): void {

exampledata.forEach((data: any) => {

if (data.artist === 'The Beatles') {

const tracks = data.tracks.map((trackData: any) => new Track(trackData.id, trackData.number, trackData.title, trackData.lyrics, trackData.video));

const album = new Album(data.id, data.title, data.artist, data.description, data.year, data.image, tracks);

this.albums.push(album);

}

});

}

/\*\*

\* The function "getArtists" returns an array of Artist objects.

\* @returns An array of Artist objects.

\*/

public getArtists(): Artist[] {

return this.artists;

}

/\*\*

\* The function "getAlbums" returns an array of albums for a given artist.

\* @param {string} artist - The `artist` parameter is a string that represents the name of an artist.

\* @returns The albums array.

\*/

public getAlbums(artist: string): Album[] {

return this.albums;

}

/\*\*

\* The function retrieves an album by its artist and ID, and returns it with its tracks if found, otherwise it returns

\* undefined.

\* @param {string} artist - The artist parameter is a string that represents the name of the artist.

\* @param {number} id - The `id` parameter is a number that represents the unique identifier of the album.

\* @returns an instance of the Album class if a matching album is found in the albums array. If no matching album is

\* found, it returns undefined.

\*/

public getAlbum(artist: string, id: number): Album | undefined {

const album = this.albums.find((a) => a.Artist === artist && a.Id === id);

if (album) {

const tracks = album.Tracks.map((track) => new Track(track.Id, track.Number, track.Title, track.Lyrics, track.Video));

return new Album(album.Id, album.Title, album.Artist, album.Description, album.Year, album.Image, tracks);

}

return undefined;

}

/\*\*

\* The createAlbum function adds a new album to the albums array.

\* @param {Album} album - The parameter "album" is of type "Album".

\*/

public createAlbum(album: Album): void {

this.albums.push(album);

}

/\*\*

\* The function updates an album in an array of albums based on its ID.

\* @param {Album} album - The `album` parameter is an object of type `Album`.

\*/

public updateAlbum(album: Album): void {

const index = this.albums.findIndex((a) => a.Id === album.Id);

if (index !== -1) {

this.albums.splice(index, 1, album);

}

}

/\*\*

\* The deleteAlbum function removes an album from an array of albums based on its id.

\* @param {number} id - The id parameter is a number that represents the unique identifier of the album that needs to be

\* deleted.

\* @param {string} artist - The "artist" parameter is a string that represents the name of the artist associated with the

\* album.

\*/

public deleteAlbum(id: number, artist: string): void {

const index = this.albums.findIndex((a) => a.Id === id);

if (index !== -1) {

this.albums.splice(index, 1);

}

}

}

**References:**

Angular. (n.d.). https://angular.io/api/forms/NgModel

Angular. (n.d.-a). https://angular.io/guide/property-binding