*Owen Lindsey*

*Professor Sparks, James*

*CST-391 Activity 3*

*10/27/2024*

**Part 1: Simple App**

*Research questions:*

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

*The @Input decorator in Angular is a way of defining inputs for a component. It allows data to flow from a parent component into a child component. Essentially, it marks a class field as an input property and supplies configuration metadata. The @Input property is bound to a DOM property in the template. When an external component interacts with the child component, it can pass data to the @Input property, allowing the child component to use external data within its class.*

*For example, in info.component.ts, if we have:*

*```typescript*

*@Input() name!: string;*

*```*

*This means the InfoComponent expects to receive a name value from its parent component, which it can use internally, such as displaying it in its template.*

**2. Describe [value] used in info.component.html:**

*The [value] binding in Angular templates is a way to bind a property of a component to the value attribute of an HTML element. This is a one-way binding from the component to the view. It is often used within `select` elements to bind each `option` element's value to a dynamic value from the component's class.*

*```html*

*<option \*ngFor="let product of products" [value]="product">{{product}}</option>*

*```*

*This code iterates over the products array defined in the component's TypeScript file and sets the value of each `option` in a `select` dropdown to the corresponding product. When a user selects an option, the selected product's value is used, which can be bound to a model or used in form submission.*

**3. Describe [(ngModel)] also used in info.component.html:**

*The [(ngModel)] directive is used in Angular forms to enable two-way data binding between the template form control and the component's property. This means that any changes to the form control's value in the UI are immediately reflected in the associated component's property and vice versa.*

*```html*

*<input type="text" [(ngModel)]="selectedProduct" name="product">*

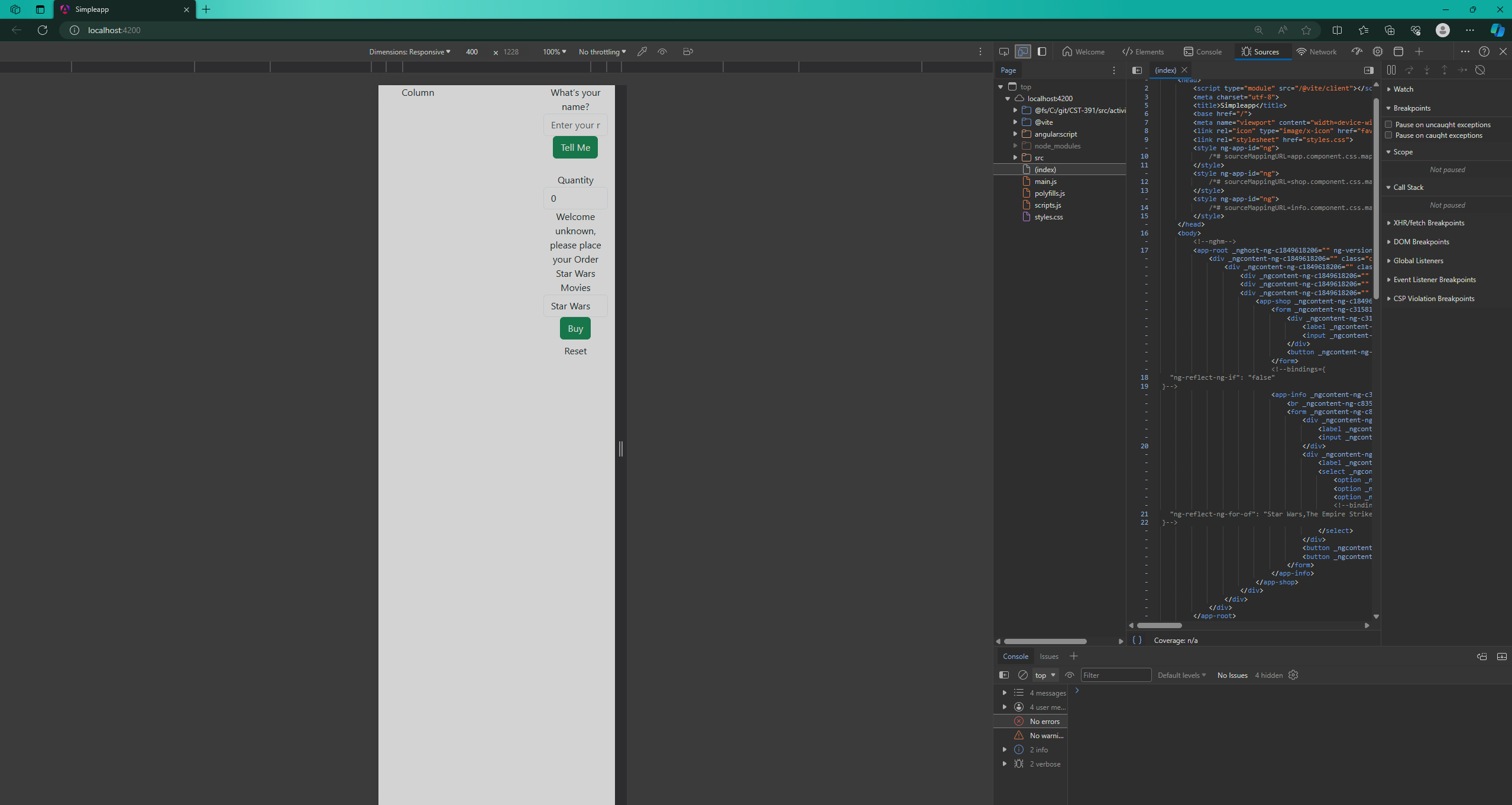
*```*

*It binds the input element's value to the selectedProduct property on the component. If the user changes the input, selectedProduct is updated with the new value. Likewise, if selectedProduct is programmatically changed in the component, the input's value in the UI is updated to reflect this change. This directive is particularly useful for forms, as it simplifies handling form inputs and validations by ensuring the view and the component are always in sync.*

**Part 1: Simple App**

*Screenshot of small screen to demonstrate the responsive grid*

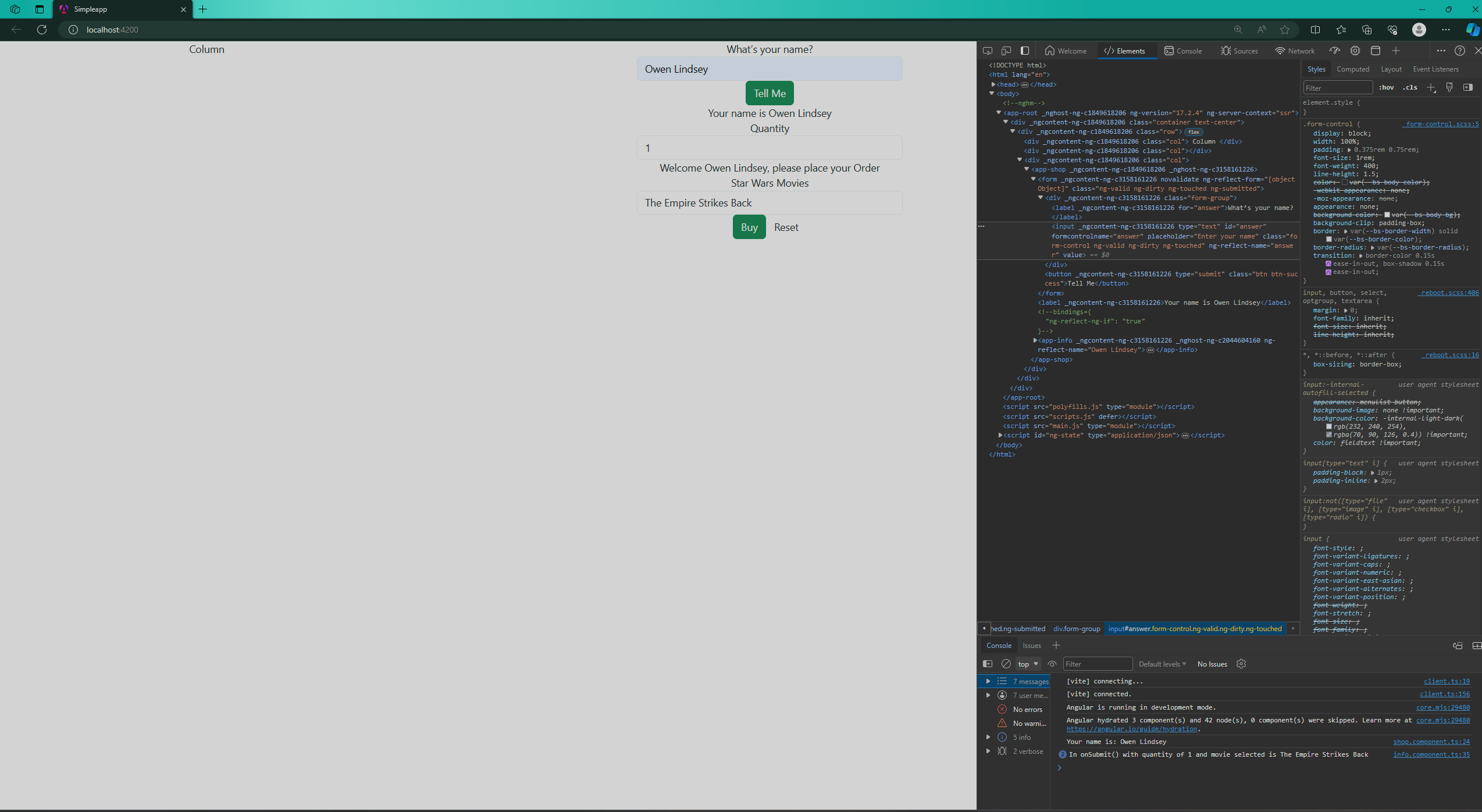
*Also shows before name is entered:*



**Part 1: Simple App**

*Screenshot of big screen to demonstrate the responsive grid*

*Shows after name is entered*

*Shows dev tools with quantity and product on the buy button:*  


Part 2: Music Application – The Front End

*Commented music-service.service.ts code:*

*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 MusicService is responsible for handling music data operations.*

*// It communicates with the backend services to fetch and manipulate music data.*

*@Injectable({ providedIn: 'root' })*

*export class MusicServiceService {*

*// An array to store artist information.*

*private readonly artists: Artist[] = [];*

*// An array to store album information.*

*private readonly albums: Album[] = [];*

*constructor() {*

*this.createArtists();*

*this.createAlbums();*

*}*

*// Initializes the artists array with predefined data.*

*private createArtists(): void {*

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

*}*

*// Reads the example data and creates albums, only including albums by 'The Beatles'.*

*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);*

*}*

*});*

*}*

*// Retrieves all artists.*

*public getArtists(): Artist[] {*

*return this.artists;*

*}*

*// Retrieves all albums for a given artist.*

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

*return this.albums;*

*}*

*// Retrieves a specific album by its artist and ID.*

Part 2: Music Application – The Front End

*Continued Commented music-service.service.ts code:*

*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;*

*}*

*// Adds a new album to the collection.*

*public createAlbum(album: Album): void {*

*this.albums.push(album);*

*}*

*// Updates an existing album's information.*

*public updateAlbum(album: Album): void {*

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

*if (index !== -1) {*

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

*}*

*}*

*/// Deletes an album from the collection based on its ID and artist.*

*// Returns 0 if successful, -1 if the album is not found.*

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

*const index = this.albums.findIndex(a => a.Id === id && a.Artist === artist);*

*if (index !== -1) {*

*// Remove the album from the albums array*

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

*return 0; // Indicate success*

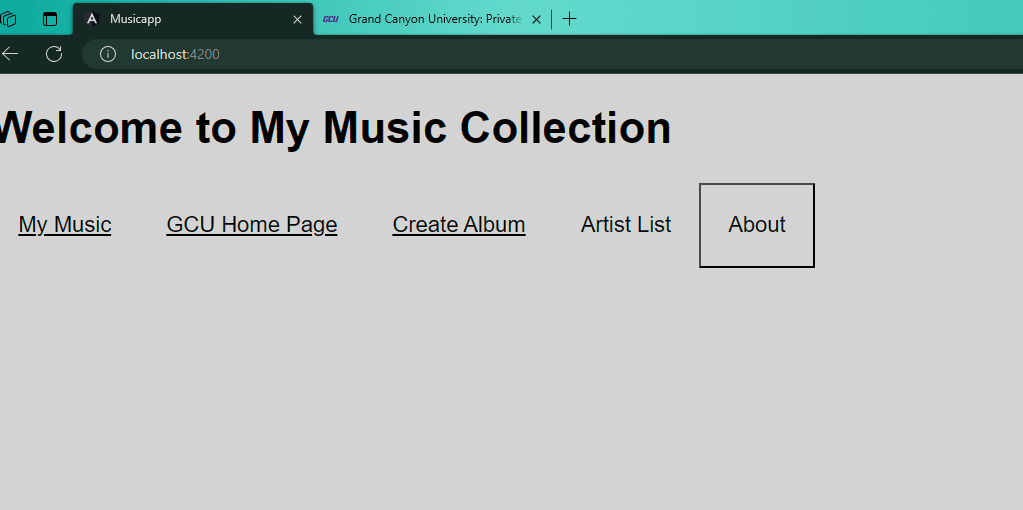
*}*

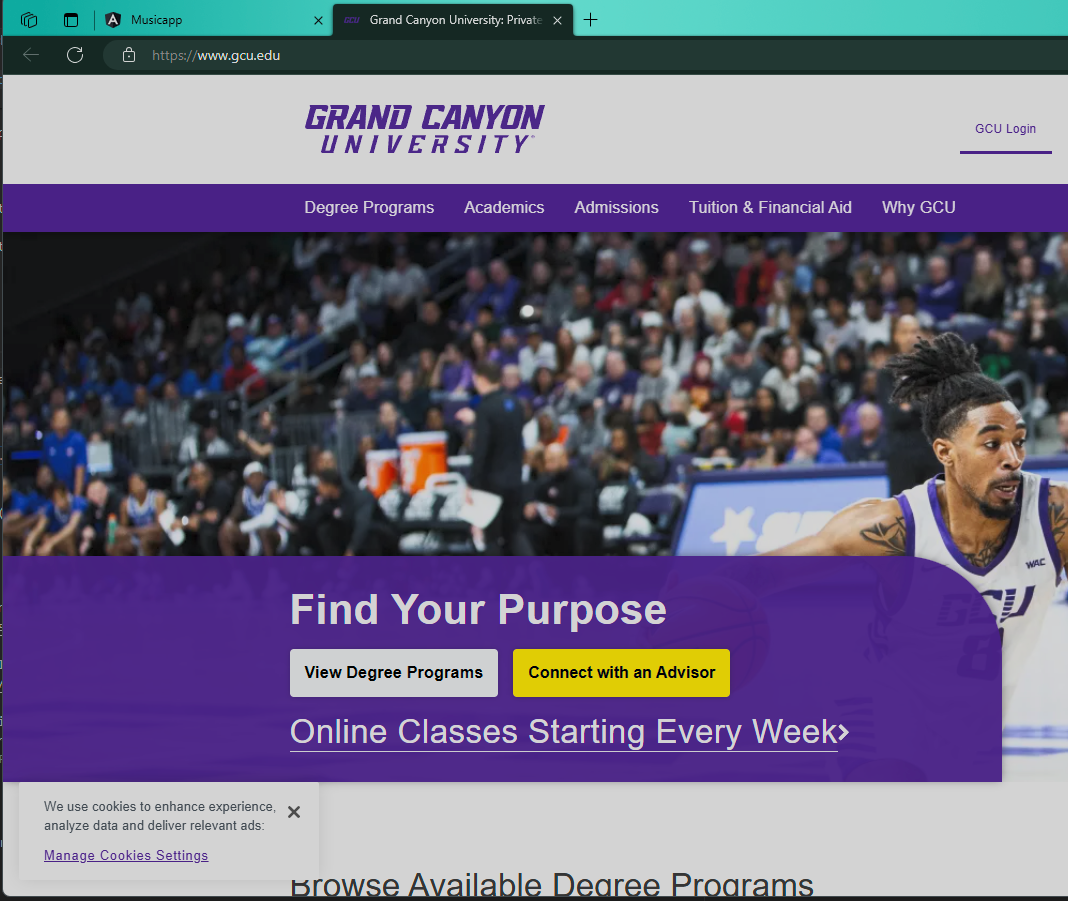
*return -1; // Indicate failure*

*}*

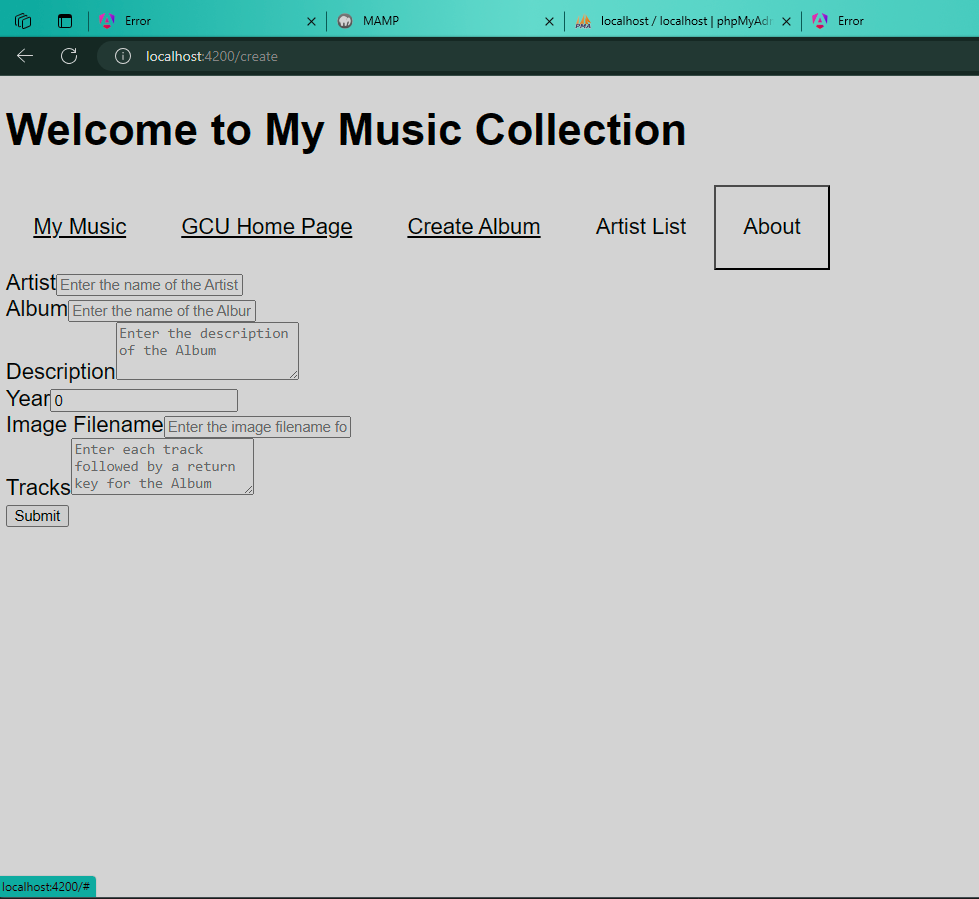
*} END FILE*

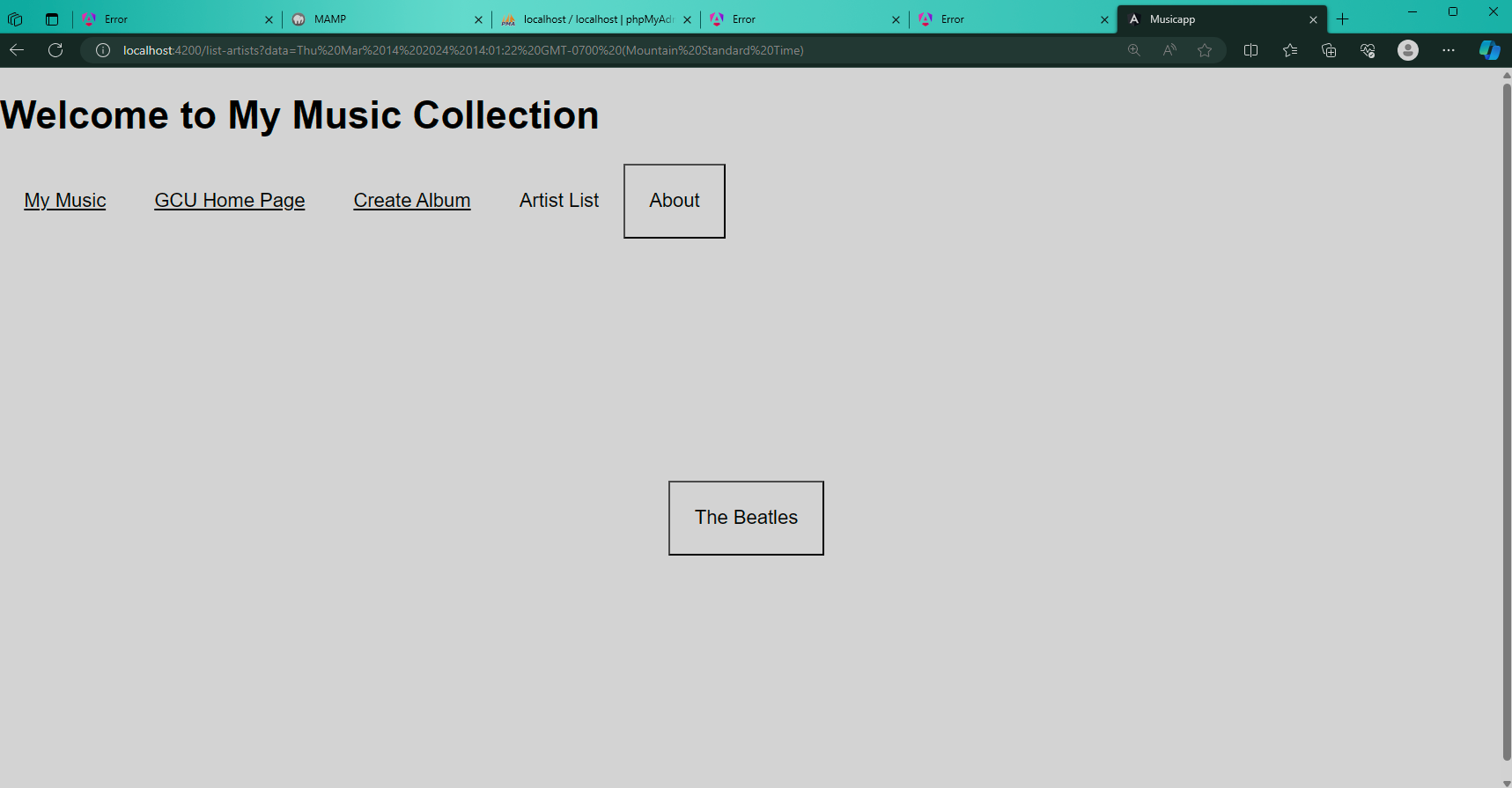
*Part 2 – Music Application – The Front End*

*Screenshot of the initial screen:*  


*Screenshot of GCU homepage:*  


*Part 2 – Music Application – The Front End*

*Screenshot of Create Album Page:*  


*Screenshot of Artist List page showing our added album/artist:*  


*Part 2 – Music Application – The Front End*

*Screenshot of About Box:*  
