**Jason Hatfield**

**Activity 3**

**CST-391**

**5/2/2023**

# Part 1

Graphical user interface, application, website

Description automatically generated

Figure 1 Small Screen

Graphical user interface, text, application

Description automatically generated

Figure 2 Large Screen

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3 No name

Graphical user interface, text, application, email

Description automatically generated

Figure 4 Name

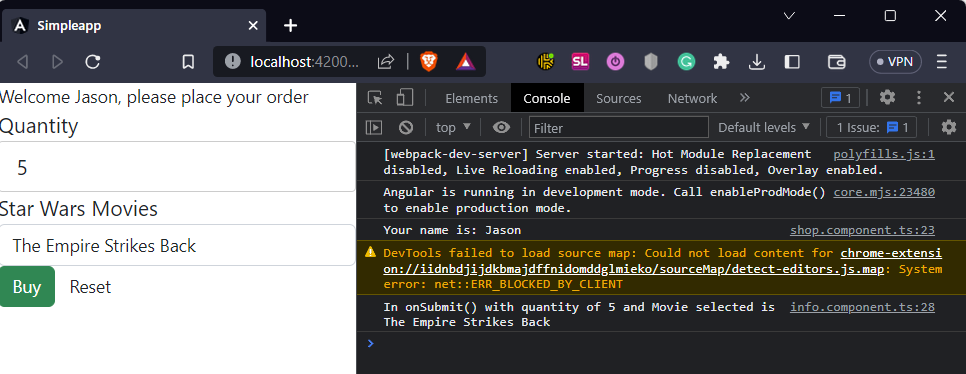


Figure 5 Browser with Dev Tools

**Research Questions**

1. In Angular, the @Input decorator passes data from a parent to a child component. A property is marked as an input property by the decorator, meaning a parent component may set it. The @Input decorator is used in the info.component.ts file to specify the property name as an input property. The info.component.html file uses this attribute to display a greeting to the user. (Angular, n.d.)
2. Angular's [value] attribute establishes a binding between a value and an input element. The [value] attribute is utilized in the info.component.html file to establish a binding between the product variable's value and the option element's value attribute. The property mentioned above establishes the default value of the select component as the value of the chosen product. (Angular, n.d.-b)
3. [(ngModel)] is an Angular two-way data binding syntax. It allows the data to flow between the view and the component. In the info.component.html file, [(ngModel)] is used to bind the quantity and selectedProduct variables to the input elements on the form. The user can alter the values of said variables via the input elements. The element can modify the variables' values in response to user-initiated modifications. (Angular, n.d.-c)

**References**

Angular. (n.d.-a). Sharing Data Between Child and Parent Directives and Components. https://angular.io/guide/inputs-outputs

Angular. (n.d.-b). Binding Syntax. https://angular.io/guide/binding-syntax

Angular. (n.d.-c). Two-way Binding. <https://angular.io/guide/two-way-binding>

# Part 2

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 6 Initial Application Page

A group of people wearing graduation gowns and caps

Description automatically generated with medium confidence

Figure 7 GCU Home Page

Graphical user interface

Description automatically generated

Figure 8 Create Album Page

Graphical user interface, text, application

Description automatically generated

Figure 9 Success

Graphical user interface, application

Description automatically generated

Figure 10 Master of Puppets Album Added

Text

Description automatically generated

Figure 11 Artist List Page Showing the added album/artist

Graphical user interface, text, chat or text message

Description automatically generated

Figure 12 About Box

/\*

\* Injectable service for managing sample music data.

\*/

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

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

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

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

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

@Injectable({providedIn: 'root'})

export class MusicServiceService {

// Define arrays to store artists and albums.

private readonly artists: Artist[] = [];

private readonly albums: Album[] = [];

// Initialize artists and albums arrays by calling createArtists() and createAlbums() methods.

constructor() {

this.createArtists();

this.createAlbums();

}

// Create artists array and add sample artist 'The Beatles'.

private createArtists(): void {

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

}

// Create albums array by parsing exampledata and adding albums where artist is '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);

}

});

}

// Get all artists in the artists array.

public getArtists(): Artist[] {

return this.artists;

}

// Get all albums in the albums array where artist is passed in as a parameter.

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

return this.albums;

}

// Get the album in the albums array where the artist and id are passed in as parameters.

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;

}

// Add a new album to the albums array.

public createAlbum(album: Album): void {

this.albums.push(album);

}

// Update an existing album in the albums array.

public updateAlbum(album: Album): void {

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

if (index !== -1) {

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

}

}

// Remove an album from the albums array where the id is passed in as a parameter.

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

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

if (index !== -1) {

this.albums.splice(index, 1);

}

}

}