CST-391 Activity 4 Guide

Contents

[Finish the Music Application – Integration with Back End 2](#_Toc126840539)

[Create a Copy of the Music Application 2](#_Toc126840540)

[Add the HTTP Client Module to the application: 2](#_Toc126840541)

[Refactor the Music Service, music-service.service.ts 2](#_Toc126840542)

[Research 7](#_Toc126840543)

[Submission 7](#_Toc126840544)

# Finish the Music Application – Integration with Back End

**Overview**

In this activity, students will finish building the Music application by integrating the application with the back end services created in Topic 1. Remember that you need to run MAMP and your Express MusicAPI to provide a live data set to your Angular front end application.

**Execution**

Execute this assignment according to the following guidelines:

### Create a Copy of the Music Application

1. Create a new directory where you want the updated Music application to be located.
2. Copy the *musicapp* (make sure to copy ALL hidden files and folders) directory from the Topic 3 Workspace to the new directory created in the previous step*.*
3. Open VS Code. Close the current Workspace.
4. Click the App Workspace Folder hyperlink from the VS Code start screen.
5. Select the *musicapp* directory where the updated music application was copied to. Refresh the Workspace by clicking the Refresh Explorer icon.
6. Select the File > Save Workspace As menu options and save your Workspace so VS Code can be opened again in the future

### Add the HTTP Client Module to the application:

1. Open the App Module *app.module.ts* file located in the src/app directory. Add the HttpClient Module import as shown below and add the Module to the imports list.

import { HttpClientModule } from '@angular/common/http';

*and*

imports: [

BrowserModule,

HttpClientModule,

FormsModule,

AppRoutingModule

],

### Refactor the Music Service, music-service.service.ts

**Remove the Hard-Coded Data and Use HttpClient to get Live Data**

1. Inject an instance of the HttpClient in the constructor.

constructor(private http: HttpClient) {}

1. As you work, remove the methods that use hard-coded data and then finally remove the hard-coded data. You should be able to run and test incrementally as you work through the methods.
2. Add a new private Http connection property (my Express MusicAPI server listens to port 5000):

Text

Description automatically generated

Figure 1 *'host' property*

1. Update the getArtists() method:
   1. Update the method signature to add a callback method and remove its return parameter:

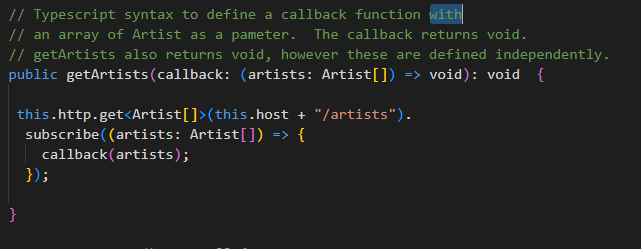


Figure 2 *Updated 'getArtists'*

* 1. Update the call in *list-artists.component.ts:*

Text

Description automatically generated

Figure 3 *Updated 'ngOnInit' Method*

* 1. You should be able to test the updated *getArtists* method in your Angular UI. Of course, its results are not compatible with the hard-coded data, but you should be able to now view a live list of artists
  2. This is a good time to think about:
     1. How the asynchronous calls are working in Angular.
     2. How these calls are similar to the asynch programming we did for the Express API. However, those methods used JavaScript promises. We can use those in Angular, but the Angular HttpClient is built around Observables. Read more [here](https://angular.io/api/common/http/HttpClient).
        1. JavaScript Promises are single-use mechanisms. An asynchronous call is made and the Promise either resolves or rejects, then it is done. Observables are not single-use mechanisms. A subscription can receive a stream of updates. This is an advantage, but forgetting to unsubscribe can cause a harmful resource leak as the subscription will continue to work in a zombie fashion after its host component is destroyed. We have no zombies in our application, the HttpClient methods automatically unsubscribe when the call is complete.
     3. Examine how callback methods are defined. The syntax makes sense, but I had to try a lot of combinations to get it to work.
     4. Understand this syntax:

*this!.artist!.artist*

* 1. Do the remaining work of replacing the hard-coded data with live API data. The complete code for *music-service.service.ts* is provided later in Figures 5 and 6*.* You must modify the calling code, replacing data structures with callback methods.
     1. Replace the methods in *music-service.service.ts* one by one.
     2. Since the method signature will change, update the calling code. Follow the pattern shown by *getArtists*.
     3. Test the updated method.
     4. Repeat steps 1 through 3 until complete.

Graphical user interface, text

Description automatically generated

Figure 4 *ngOnInit Update*

A tricky call: *list-albums.component.ts*, the other service calls are left to the reader.

Text

Description automatically generated

Figure 5 *Part 1:* *music-service.service.ts*

Text

Description automatically generated

Figure 6 *Part 2:* *music-service.service.ts*

1. Test final application:
2. Take captioned screenshots for the following:
3. Main Application screen
4. Artist List screen,
5. Album List screen
6. Album Display (with tracks) screen
7. Add Album screen
8. Edit Album screen
   * + 1. Optional
9. The Delete Album (response) screen
   * + 1. Optional

### Research

1. Research how an Angular application maintains a logged in state. How does it communicate this state to the server?

### Submission

Submit the following in a Microsoft Word document as directed by the instructor:

**Deliverables**

1. All captioned screenshots.
2. Research question answered.