WEB422 Assignment 4

Submission Deadline:

Friday, March 11th 2022 @ 11:59pm

Assessment Weight:

9% of your final course Grade

Objective:

For this assignment, we will be working with data from Spotify to create a User Interface (UI) with Angular / Angular Material that enables users to explore and discover new music. We will not be using "live" data for this assignment, however it will provide the foundation for the assignments for the remainder of the semester.

Sample Solution:

You can interact with a working version of the assignment at the following link:

https://pedantic-galileo-41edc0.netlify.app

Step 1: Creating an Angular App & Downloading the Static data

To begin this project we must create a new Angular app using the @angular/cli command line tool: "ng". If you do not have it installed yet, you can pull it from npm using the command:

npm install -g @angular/cli

Once the Angular CLI has been installed, you should be able to kickstart a new Angular application with the following command (**Note:** The options here enable routing, skip testing and skip the initialization of a "git" repository, respectfully)

ng new web422-a4 --routing -S -g

Before development of the app can begin, we need to make sure we have access to the static data. As mentioned above, we will not need to create an account / application with Spotify for this assignment. Instead, the following queries have been completed for you. For each of the queries, download the "result" and place the .json file a **new "data" directory** located in "**src/app/data**" (ie: src/app/data/*someFile*.json):

- https://developer.spotify.com/documentation/web-api/reference/#/operations/get-new-releases
    - Result (Static list of new Releases): https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/data/NewReleasesAlbums.json

- https://developer.spotify.com/documentation/web-api/reference/#/operations/get-an-artists-albums
  (see: Get an Artist's Albums)
    - Result (Albums for the artist U2):
      https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/data/SearchResultsAlbums.json

- https://developer.spotify.com/documentation/web-api/reference/#/operations/get-an-album
  (see: Get an Album)
    - Result (Get a Specific U2 Album: "The Joshua Tree"):
      https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/data/SearchResultsAlbum.json

- https://developer.spotify.com/documentation/web-api/reference/#/operations/get-an-artist
  (see: Get an Artist)
    - Result (the artist U2):
      https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/data/SearchResultsArtist.json

Once you have obtained all 4 of the above JSON files, your **src/app/data** folder should now contain:

- NewReleasesAlbums.json
- SearchResultsAlbums.json
- SearchResultsAlbum.json
- SearchResultsArtist.json

In order to allow our components to read from these files, we need to add the properties:

"allowSyntheticDefaultImports": true

and

"resolveJsonModule": true,
"esModuleInterop": true

to the "**tsconfig.json**" file, in the below locations, ie:

```
/* To learn more about this file see: https://angular.io/config/tsconfig. */
{
  "compileOnSave": false,
  "allowSyntheticDefaultImports": true,
  "compilerOptions": {
    "resolveJsonModule": true,
    "esModuleInterop": true,
     ...
  }
}
```

Next, we need to obtain the UI components that we will use for this application. Since we're using Angular, we will leverage "Angular Material" and "Angular Flex-Layout"

To begin, let's include Angular Material. You can do this by executing the command "**ng add @angular/material**" in the terminal (ensure that you're in your "web422-a4" directory).

- When prompted: "Choose a prebuilt theme name, or "custom" for a custom theme:", feel free to choose **any** one. If you wish to recreate the sample, the first option "Indigo / Pink" was used.
- Set up global Angular Material typography styles? – choose **Y**
- Set up browser animations for Angular Material? – choose **Y**

Next, we need to add Angular Flex Layout:

- Enter the command "**npm i @angular/flex-layout**" to install it.

Now, if we wish to make use of the functionality provided by Angular Material / Angular Flex-Layout, we need to import some modules into our "**src/app/app.modules.ts**" file, specifically:

```
import { MatIconModule } from '@angular/material/icon';
import { MatSidenavModule } from '@angular/material/sidenav';
import { MatProgressBarModule } from '@angular/material/progress-bar';
import { MatMenuModule } from '@angular/material/menu';
import { MatToolbarModule } from '@angular/material/toolbar';
import { MatListModule } from '@angular/material/list';
import { MatButtonModule } from '@angular/material/button';
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatCardModule } from '@angular/material/card';
import { MatChipsModule } from '@angular/material/chips';
import { FlexLayoutModule } from '@angular/flex-layout';
```

Once you have pasted the above import statements, you must add the imported modules to the "imports" array, in the @NgModule({ … }) decorator:

```
MatIconModule,
MatSidenavModule,
MatProgressBarModule,
MatMenuModule,
MatToolbarModule,
MatListModule,
MatButtonModule,
MatFormFieldModule,
MatInputModule,
MatCardModule,
MatChipsModule,
FlexLayoutModule
```

This will give you access to the components / functionality required to recreate the type of layout used in the sample. If you wish to add additional components from Angular Material: https://material.angular.io/components/categories you will have to add the corresponding import statements. These can be found by clicking on the "API" tab when viewing a component in detail.

Step 2: Adding CSS & Creating the "View" Components

Before we start creating our primary "View" components, we must delete the contents of the "**src/styles.css**" file and replace it with some custom styles instead:

https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/styles.css.txt

NOTE: If you decided to use a prebuilt theme *other* than Indigo / Pink, you will have to change the @Import statement at the top of the file to reflect your choice.

Updating app.component.html

Now that our modules and styles are in place, we should update our main **app.component.html** template to provide a consistent layout container for all of our main "views" within the application. You can grab the starter file from here:

https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/app.component.html.txt

You will note that its missing the routing features – these will have to be added manually, ie:

- The "New Releases" link in the sidebar must link to the to the "/newReleases" route
- The "About" link in the sidebar must link to the "/about" route

Additionally, there is a space for a logged in user at the top right. Since we currently do not have the functionality to register / log in users, please **change User Name to your full name** ie: "User Name" to "John Doe", etc.

Note: Once you have pasted the above code, you should be able to test your application and see the main layout rendered correctly in the browser!

Creating new "view" Components

Next, we must create the primary components that will be rendered using client side routing. These can be thought of as the "view" components.

For the next steps, use the Angular CLI (ng) to create the following components according to their individual specifications:

AboutComponent (about.component.ts)

- The purpose of this component is simply to show a relevant heading, ie: **About** – All about me … and a few paragraphs, images, etc. about yourself. You may reuse the content from your Assignment 3 if you choose.
- If you wish to test the component at this time, you can place the **<app-about></app-about>** component in the **<!-- Main content -->** section of the **app.component.html** file to see the result

NotFoundComponent (not-found.component.ts)

- The purpose of this component is simply to show a relevant heading, ie: **Resource Not Found** and some message / image, animation, etc. informing the user that the app cannot fulfill the request – this is the "404 Page" of the application.

- If you wish to test the component at this time, you can place the **<app-not-found></app-not-found>** component in the **<!-- Main content -->** section of the **app.component.html** file to see the result

NewReleasesComponent (new-releases.component.ts)

- The purpose of this component is to render all of the "new releases" from our "newReleasesAlbums.json" file. To obtain the contents of this file, we must:
    - Add the following "import" statement:
      import data from '../data/NewReleasesAlbums.json';
    - Create a "releases" property within the class
    - When the component is *Initialized* set the value of the "releases" property to:
      data.albums.items

- In the template for the component be sure to render a relevant static heading informing the user that they're currently viewing the "**New Releases**" and that they can click on either the "**album cover**" or "**artist**" name for more information

- For the data displayed, you can use the following static HTML code as a starting point for your output – This renders a single "card" showing one of potentially many albums

  NOTE: Your solution will need to render the data from the "releases" property instead of the included static data

  https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/new-releases-component.html.txt

- The "Album Cover" image must link to the "/album" route
- The "Artist Selection" chips must each link to the "/artist" route

- If you wish to test the component at this time, you can place the **<app-new-releases></app-new-releases>** component in the **<!-- Main content -->** section of the **app.component.html** file to see the result

AlbumComponent (album.component.ts)

- The purpose of this component is to render the "album" from our "searchResultsAlbum.json" file. To obtain the contents of this file, we must:
    - Add the following "import" statement:
      import albumData from '../data/SearchResultsAlbum.json';
    - Create an "album" property within the class
    - When the component is *Initialized* set the value of the "album" property to: **albumData**
- In the template for the component be sure to render a relevant heading informing the user which "***Album Name***" they're viewing (where ***Album Name*** is the current album)

- For the data displayed, you can use the following static HTML code as a starting point for your output – This renders a single "card" containing album information with a "track listing" section showing one of potentially many tracks.

NOTE: Your solution will need to render the data from the "album" property instead of the included static data

https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/album-component.html.txt

- The "Artist Selection" chips must each link to the "/artist" route
- If you wish to test the component at this time, you can place the **<app-album></app-album>** component in the **<!-- Main content -->** section of the **app.component.html** file to see the result

ArtistDiscographyComponent (artist-discography.component.ts)

- The purpose of this component is to render information for a specific "artist" from our "searchResultsArtist.json" file, as well as a list of albums from our "searchResultsAlbums" file . To obtain the contents of these files, we must:
    - Add the following "import" statements:
      import albumData from '../data/SearchResultsAlbums.json';
      import artistData from '../data/SearchResultsArtist.json';
    - Create an "albums" property within the class
    - Create an "artist" property within the class
    - When the component is *Initialized* set the value of the "albums" property to:
      albumData.items.filter((curValue, index, self) => self.findIndex(t => t.name.toUpperCase() === curValue.name.toUpperCase()) === index);

      NOTE: This is to filter out duplicate album names. You may use your own logic here, if you wish.
    - Also, set the value of the "artist" property to: **artistData;**
- In the template for the component be sure to render a relevant heading informing the user which "***Artist Name***" they're viewing (where ***Artist Name*** is the current artist) as well as reminding them that they can click the "**album cover**" for more information
- For the data displayed, you can use the following static HTML code as a starting point for your output – This renders a single "card" showing the artist image as well as a second single "card" showing one of potentially many albums

  NOTE: Your solution will need to render the data from the "artist" and "albums" properties instead of the included static data

  https://pat-crawford-sdds.netlify.app/shared/winter-2022/web422/A4/artist-discography-component.html.txt
- The "Album Cover" image must link to the "/album" route

- If you wish to test the component at this time, you can place the **<app-artist-discography></app-artist-discography>** component in the **<!-- Main content -->** section of the **app.component.html** file to see the result

Step 3: Main Route Configuration

Now that our main components are in place, we can set up our routing configuration. To begin, configure the following routes in your **app-routing.module.ts** file by adding the configurations to the **Routes** array:

- **Path: "newReleases" -** Shows the **NewReleasesComponent**
- **Path: "artist" -** Shows the **ArtistDiscographyComponent**
- **Path: "album" -** Shows the **AlbumComponent**
- **Path: "about"** – Shows the **AboutComponent**
- **Path: "" -** Redirects to the "**/newReleases**" Route
- **No Route Found -** Shows the **NotFoundComponent**

Once this is complete, place the "router outlet" component **(<router-outlet></router-outlet>)** in the **<!-- Main content -->** section of the **app.component.html** file

Assignment Submission:

- For this assignment, you will be required to **build** your assignment before submitting. This will involve running the command:
    - **ng build**

  to create a "dist" folder that you will include in your submission
- Add the following declaration at the top of your app.component.ts file:

```
/*********************************************************************************
* WEB422 – Assignment 04
* I declare that this assignment is my own work in accordance with Seneca  Academic Policy.  No part of this
* assignment has been copied manually or electronically from any other source (including web sites) or
* distributed to other students.
*
* Name: _____ Student ID: _____ Date: _____
*
*********************************************************************************/
```

- Compress (.zip) the all files in your Visual Studio code folder **EXCEPT the node_modules folder AND the .angular folder** (this will just make your submission unnecessarily large, and all your module dependencies should be in your package.json file anyway).
- Submit your compressed file (without the node_modules folder) to My.Seneca under **Assignments** -> **Assignment** 4

Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.