A3: Spotify Browser in Angular

Overview

In this assignment, you'll demonstrate your ability to (1) gather data from an API, and (2) develop a frontend interface which displays the data. You'll use Angular to create a webpage which communicates with a backend server written in Node.js/Express to browse the music on Spotify, a popular music streaming app. The provided backend server will handle authentication via OAuth and communicate with the Spotify API, but you will have to make HTTP requests to the backend server to trigger API calls. Your page will support searching for artists, albums, and tracks and navigating between the three resources.

Assignment Details

Starter code

A starter repository is on GitHub Classroom.

Repository Structure

The repository contains somewhere around 80 files. You will not need to edit most of them in this assignment. The repository contains two folders: `webserver` and `client`. `webserver` includes the Express/Node.js backend for communicating with the Spotify API. `client` includes the Angular frontend for browsing music.

In the backend (`webserver` folder), you will not need to edit any of the JavaScript files; they are provided for you. However, you do need to create two files:

1. client_secret.json
2. tokens.json

Both files contain secret information, so they should not be committed to the repository and are therefore listed in the .gitignore file. (Which is also why we cannot provide them to you)

In the frontend (`client folder`), the files which need editing are all under the `app` folder. Within that folder, the files/subfolders you should edit are:

- The six components in the components folder: `about`, `carousel`, `carousel-card`, `search`, `track-list`, and `thermometer`. Each component folder contains four files (`*.component.css`, `*.component.html`, `*.component.ts` , `*.component.spec.ts`). Of these files, all edits will be made in the `.component.html` and `.component.ts` files.
- Three of the four components in the pages folder: `album-page`, `artist-page`, and `track-page`. `home-page` will not be edited. Again, all edits will be made in the `.component.html` and `.component.ts` files.
- The one service in the services folder: `spotify-service`. All edits will be made in the `.service.ts` file, none in the `.service.spec.ts` file.
- One of the six classes in the data folder: `track-features.ts`. The four other classes do not need to be edited.

You will also edit the readme.txt file in the root folder.

This assignment can be completed without writing any additional files or functions. But you may find it helpful to add a file or function, such as to complete one of the bonus features.

Setting up your Workspace

If you're feeling unfamiliar with the command line, you might find the guides in the resources page helpful.

Some of the packages/libraries depend on newer versions of Node JS. Check what version of Node JS you have installed with:

```
node --version
```

If you installed Node JS before this class, make sure you've updated to at least 10, such as with nvm or downloading the latest version from the node website. If you installed Node during A2, you should not need to update, but it's probably worth checking the version number anyways.

There are no global dependencies required to run the Express webserver.

For the client, you will need to install Angular through npm. To do this, you will install the Angular Command Line Interface (CLI) globally with:

```
npm install -g @angular/cli
```

The Angular project is already set up in the `client` folder, but the CLI is necessary to run the project.

You will also need to install the dependencies for both the `webserver` and the `client`. These dependencies are defined in each project's respective `package.json` files. cd into each respective project's folder and install the dependencies with:

```
npm install
```

You may encounter a few installation warnings or high severity security vulnerabilities. We will ignore them for this assignment, but these should typically be addressed when using libraries.

When installing, if you run into issues which look like "permission denied, try changing the permissions of the node_modules directory. If you are still having trouble, ask on Zulip or talk to the course staff. It's important to try to get the setup working sooner rather than later, even if you do not plan on working on the assignment until close to the deadline.

You may get a warning that the version of Angular that you have installed is newer than the version of the project. You can safely ignore that in this assignment.

It's likely that your project code will have out of date dependencies, so before you move to the next section, cd into both the `client` and `webserver` directories and run the following command:

```
npm update
```

Running the Webserver

A Spotify Developer account is required to set up the Express webserver. Create a Spotify account or log in at https://developer.spotify.com/dashboard/ and follow the instructions to create a client id. Name the app whatever you'd like. For the App description, enter:

> This App is used to create a music browser as part of a course assignment for IN4MATX 133, User Interface Software, at the University of California, Irvine.

Indicate that you are building a Website and are not developing a commercial application.

Once you have created your application, be sure to set the redirect URI to:

```
http://localhost:8888/callback
```

This will tell Spotify to redirect back to our Express webserver once authentication and authorization is complete. You should also create a file in the `webserver` folder (not the routes folder), `client_secret.json`, with your consumer key and secret. It should be of the form:

```
{
 "client_id": "Your Client Key",
 "client_secret": "Your Client Secret"
}
```

At this time, also create a placeholder file in the same folder for `tokens.json`. This file will be overwritten once an access and refresh token have been retrieved. Your `tokens.json` file should be exactly:

```
{
 "access_token": null,
 "refresh_token": null
}
```

To run the Express webserver, cd into the webserver folder and run: npm start

This will start the webserver at:

```
localhost:8888
```

Be sure the dependencies have been installed first via:

```
npm install
```

If you make a change to code in the webserver (which you should not need to do to complete the required portions of this assignment), you will have to end the running program and re-run it. End it by typing `Ctrl-C` into the command line and then re-running it with:

```
npm start
```

Running the client

To run the Angular client, `cd` into the client folder and run:

```
ng serve --open

NOTE: '--open' is optional
```

This will start the client at `localhost:4200`. Adding the `--open` flag will open it up in the browser. Be sure the dependencies have been installed first via `npm install`.

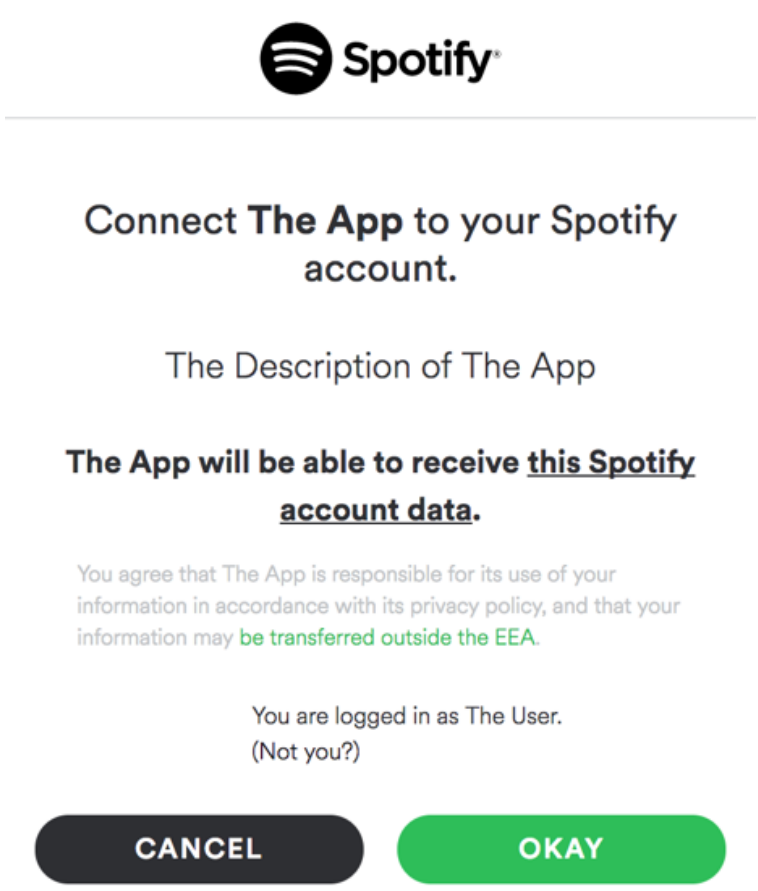
Any code changes will be automatically reloaded on the client.

Requirements

There are five parts to this assignment. The later parts use many of the same components, so once those are implemented the later parts become a matter of gluing them together. You can also optionally add one of a number of bonus features.

Part 1: Communication with the Webserver (2 point)

The first time you push the "login" button in the top-right corner of the Spotify browser, you will be redirected to Spotify to authenticate and authorize access to your app. The OAuth flow between the Spotify server and the webserver has been implemented for you. That means that once you authenticate and authorize access, you can query the webserver to make API requests on your behalf. The webserver will also handle refreshing the access token when it expires. Hooray!



However, in order to make API requests, you will need to add some functionality to the Spotify Service (`services/spotify.service.ts`). First, update the `sendRequestToExpress` function in the Spotify service to ask the Express webserver to make an API request to Spotify at the endpoint passed in. The endpoints should then return the appropriate data type for each API call. You will be able to test your function by building out the pages in the assignment.

> **Helper Clip**
>
> Watch a quick overview on sendRequestToExpress

Part 2: Spotify Browser Home Page (5 points)

In this part, you will create a homepage to view some information about the logged in user's account. You will also create search requests to the Spotify API. As you get started with Angular, it may be helpful to review the code from the demos or the lecture recordings.

Populating Information About the User (2 point)

Once API requests can be made via the Express server (Part 1), edit the `about` component to call the endpoint created for the "about me" page when the load info button is clicked. To do so, you will need to inject the Spotify service you just edited. The Spotify API endpoint reference describes what this endpoint and other endpoints return. Update the variables in the component's `.ts` file and bind them to the appropriate places in the `.html` file. This will populate the left side of the home page with the logged in user's name, profile picture, and a link to open their profile on Spotify.com.

Populating the Search Component (3 points)

The right side of the page contains the `search` component. The search component contains an input and a select (dropdown) menu to declare what to search for. Bind the values in the component's model to the input and select controls. Bind the search button to a function which uses the Spotify service to have Express make the API request. You can also search online to learn how to trigger a search when a user presses "enter" in the input field or changes the dropdown menu. These two features are optional–you are only required to bind the search button. For simplicity, your code can assume that any search typed in is valid (e.g., returns at least one artist/track/album).

Map the response from Express into an array of the appropriate resource type–ArtistData, AlbumData, or TrackData. To do so, you will have to update the Spotify Service to create objects based on the category being searched for. Similar to the example given for parsing ProfileData, the constructor of each resource can automatically parse the dictionary values Spotify returns into an Object. For example, AlbumData loads the genres of the Album and creates an array with each of the artists on the album. Note that all three of these resource classes extend the ResourceData class, which parses some dictionary values common across all the resource types. You should not need to edit any of these four files; contact the course staff if a response parses differently than you would expect or causes an error.

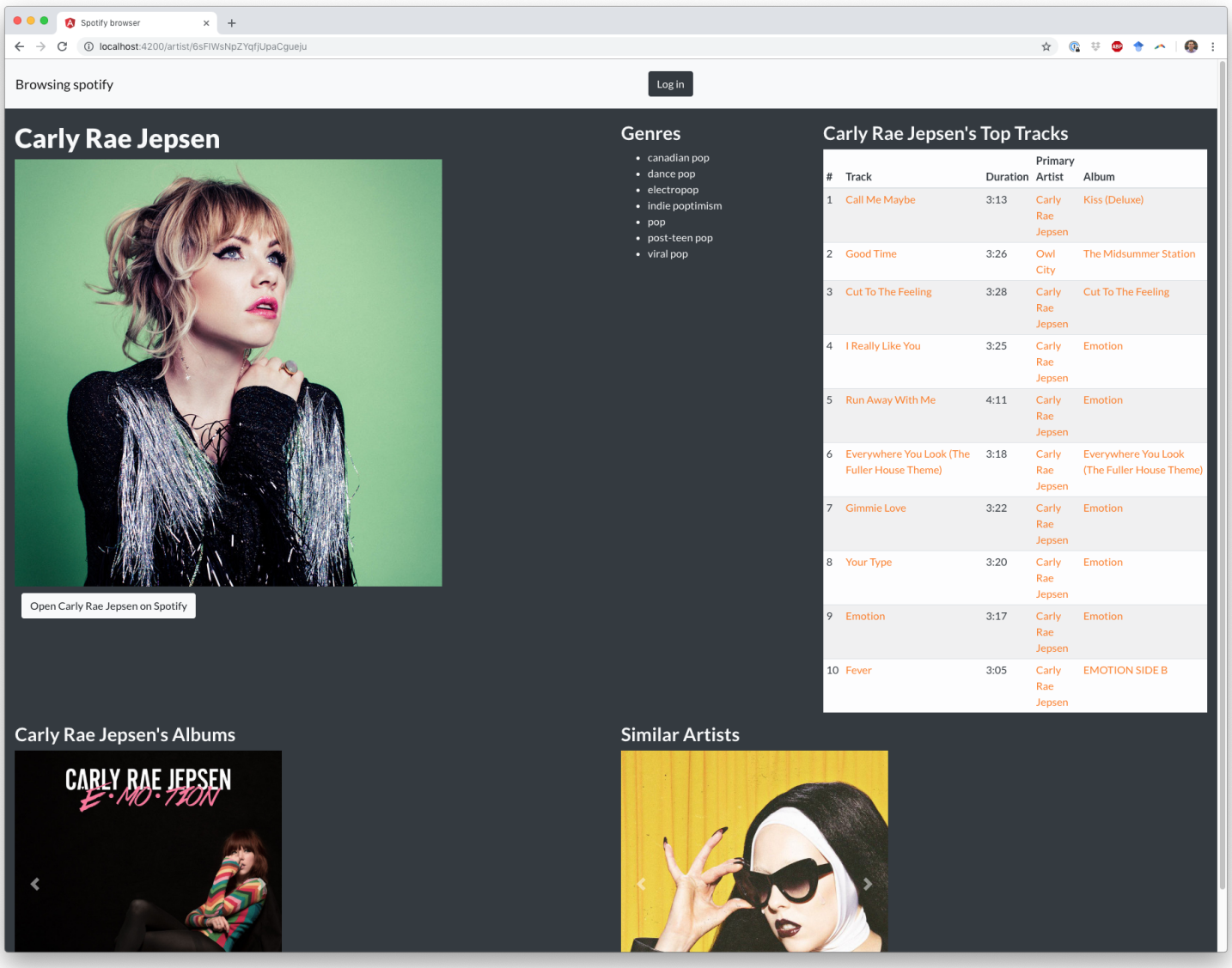
Once the array of resources has been created, display the search results in either a `carousel component` (for artists or albums) or a `track-list` component (for tracks).

The carousel is implemented via a Bootstrap carousel. Edit the carousel component to create `carousel-card` components for each resource passed in. Additionally, edit the carousel-card component to link to the local URL for browsing the specific artist or album. Check the app router (app-routing.module.ts) to identify what the URL should be.

The track list is implemented in the `track-list` component. Edit the component to display each track in the response as a row in the component's table, displaying the track's index, name, duration, artist, and album. The component should support hiding the artist and/or album columns for the track when the hideArtist or hideAlbum variables are true.

Part 3: Spotify Browser Artist Page (3 points)

In this part, you will create a page to view some information about a searched-for or linked-to artist. You will update the `artist-page` component to do this. For simplicity, you can assume the id passed via the URL is always valid (e.g., always refers to an artist which exists in the Spotify API). The artist-page includes code for retrieving the artist's id from the URL.

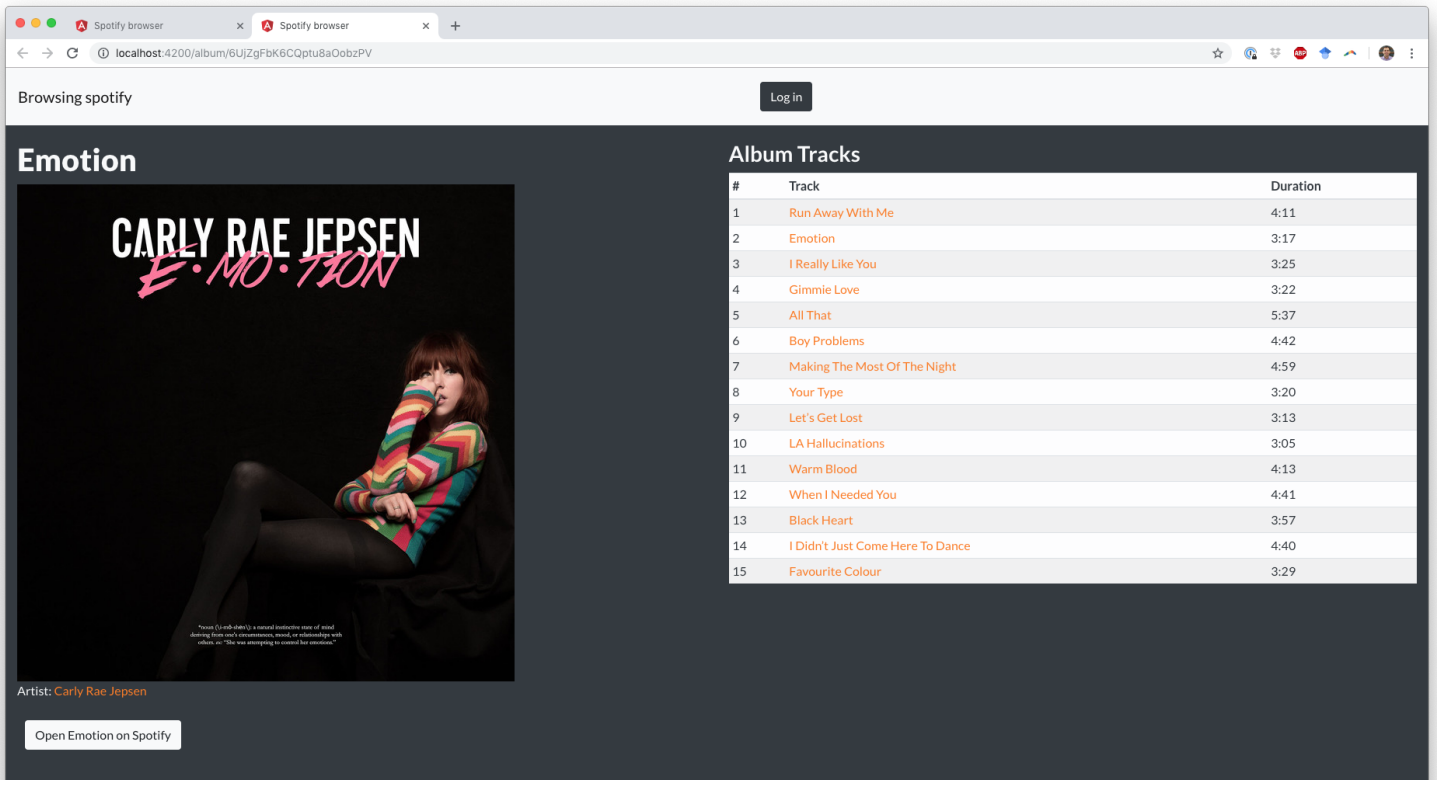


Update the Artist Page to retrieve the basic information about the artist, their top tracks, their albums, and similar artists. The Express webserver has endpoints for each of these four requests. Update the component to display the information as described in the component's template. You will re-use the `track-list` and `carousel` components on this page.

You will again have to update endpoints in the Spotify Service to get information about an artist, their top tracks, albums, etc.

Part 4: Spotify Browser Album Page (2.5 points)

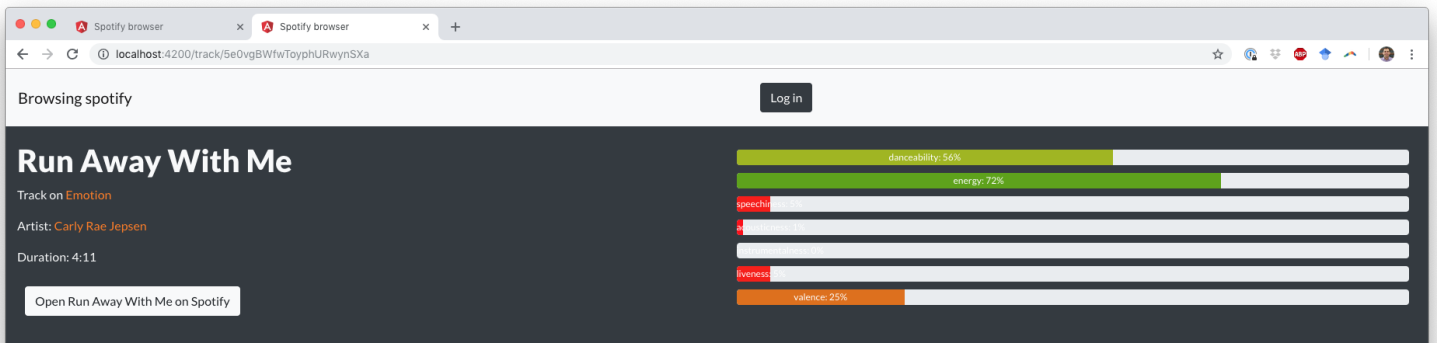
In this part, you will create a page to view some information about a searched-for or lined-to album. You will update the `album-page` component to do this. For simplicity, you can assume the id passed via the URL is always valid (e.g., always refers to an album which exists in the Spotify API). The album-page includes code for retrieving the album's id from the URL.



Update the Album Page to retrieve the basic information about the album and it's tracks. The Express webserver has endpoints for each of these two requests, which you will again need to call in the Spotify service and parse data from. Update the component to display the information as described in the component's template. You will re-use the `track-list` component on this page.

Part 5: Spotify Browser Track page (2.5 points)

In this part, you will create a page to view some information about a searched-for or lined-to track. You will update the `track-page` component to do this. For simplicity, you can assume the id passed via the URL is always valid (e.g., always refers to a track which exists in the Spotify API). The track-page includes code for retrieving the track's id from the URL.



Update the Track Page to retrieve the basic information about the track and it's audio features. The Express webserver has endpoints for each of these two requests. Update the component to display the information as described in the component's template.

For this page, you will have to edit the `thermometer` component. The thermometer uses a Bootstrap progress bar to display the track's audio features. Note the component does not have any Input fields, so you will have to identify what fields you need and define them.

You can edit the `track-feature` class to use the chroma library to display the progress bar's color as a mix between two values depending on it's percent. The provided mixes between red and green. You can choose whatever colors you like, so long as differences are visually distinct. HSL blending is recommended, as it more closely models how human vision blends colors.

Optional/Bonus Features

The five parts of the assignment create a tool for browsing the Spotify API. There are many ways of extending this browser, and we can offer one point of extra credit for implementing an extension. We've enumerated some potential extensions:

- Moving the Express webserver to a publicly visible URL rather than running on localhost. One way of doing this would be to create a Heroku instance. Students can get a free Hobby-tier server on Heroku for two years. This guide may be helpful for deploying your heroku app.
- The webserver for this assignment can only handle a single user because tokens are being saved in a JSON file with no user information. Use session middleware and an improved scheme for storing tokens to handle multiple users.
- Adding a component which plays music for an Artist/Album/Track using Spotify's Web Playback SDK. You can add the player to any page within your Spotify browser.

We are open to other suggestions for bonus features.

Submitting

To submit, zip your repository and upload it to Canvas. Any late uploads are subject to the course's late policy.

Please update your readme.txt with how long the assignment took, who or what online resources you consulted with, any bonus features you added, and anything else we should know.

Grading

This assignment will be graded on a scale of 15 points, broken down as follows:

- Communication with the Webserver (2 points)
- Spotify Browser Home Page (5 points)
- Spotify Browser Artist Page (3 points)
- Spotify Browser Album Page (2.5 points)
- Spotify Browser Track Page (2.5 points)

A bonus feature completed will earn you 1 point of extra credit (e.g., no points given for beyond one bonus feature). The maximum grade for this assignment is therefore 16/15.

In prior courses, you've been asked to follow good principles for indentation, naming variables, commenting, etc. We expect you to do the same in this course, but aim to avoid being draconian in our enforcement of these principles. Egregiously poor formatting, completely uncommented code, etc. may incur a small penalty (e.g., -1 point), but we expect this to be rarely applied.

Previous
‹ A2: Runkeeper Tweet Report in JavaScript and TypeScript

Next
A4: Sleep Tracker in Ionic ›

By Mark S. Baldwin