Spotify Streamer Implementation Guide

Spotify Streamer, Stage 1: Implementation Guide

Stage 1: Overview

This document should be used as a guide and development plan for Stage 1 of the Spotify Streamer app. It breaks down the build process into smaller, concrete milestones, and provides technical guidance on specific tasks.

Evaluation

Stage 1 is evaluated against the following rubric.

User Experience Design

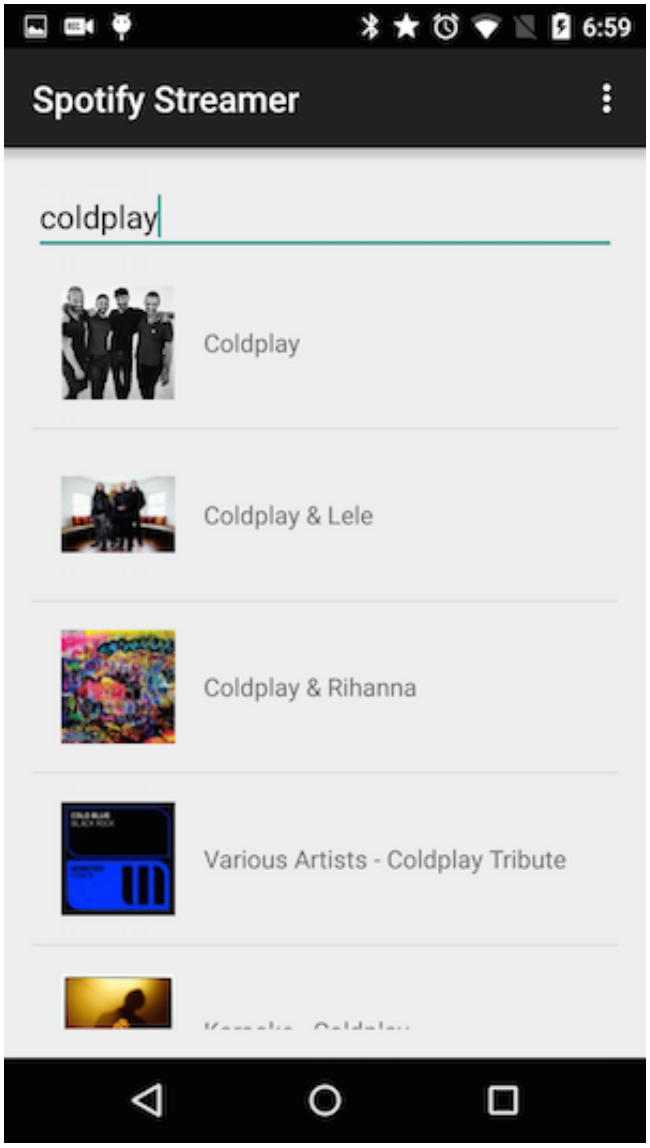
This guide will provide UI mock-ups (or mocks) to establish a baseline for Spotify Streamer's core user experience. Your job is to implement these same experiences, although you're free to customize the visual design to make it your own.

Note: In a real world situation, such mocks are generally provided by a designer or design team. But, in some cases, an individual developer creates their own mocks.

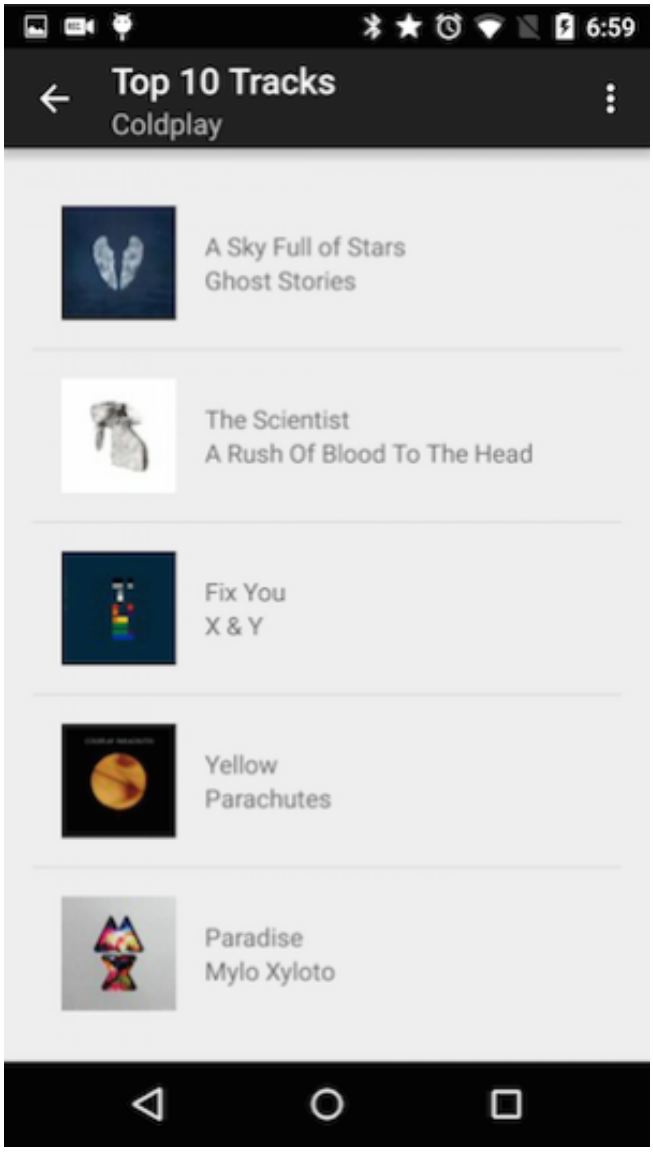
The main purpose of these mocks is to communicate and establish the visual and interactive experience of the entire app. The first two screens will be implemented in Stage 1, and the track player will be implemented in Stage 2.

Phone Interaction Flow

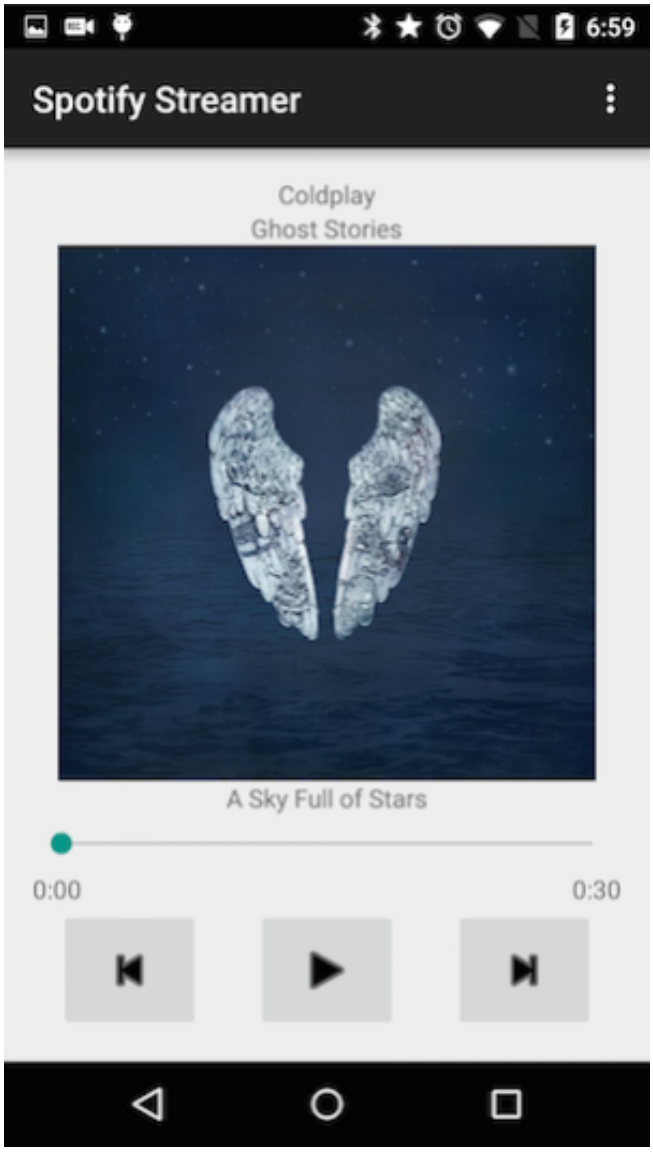
1. Search for an artist and display results



2. Then display the top tracks for a selected artist.



3. When a user selects a track, the track should start playing in a player UI (To be implemented in Stage 2)



Project Setup: Configure Your Project to Use Two Important Libraries

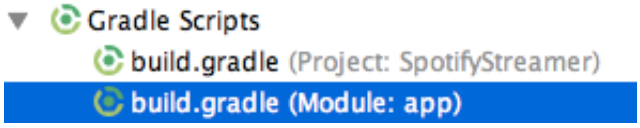
This project will utilize two key Android libraries, which we've included to reduce unnecessary extra work and help you focus on applying your app development skills.

1. Picasso - A powerful library that will handle image loading and caching on your behalf.
    a. You are welcome to use a different image loading library if you'd like. For example, Glide is another image library you can use. In this implementation guide, we'll be covering how to set up and use Picasso.
2. Spotify-Web-Api-Wrapper - A clever Java wrapper for the Spotify Web API that provides java methods that map directly to Web API endpoints. In other words, this wrapper handles making the HTTP request to a desired API endpoint, and deserializes the JSON response so you don't need to write any parsing code by hand. When you need to make a particular Web API call, you simply call the corresponding method in the library and act on

the return data, already converted to Java objects that you can interact with programmatically.

Library Configuration Instructions

You'll need to modify the build.gradle file for your app. These modifications will happen in the build.gradle file for your module's directory, *not* the project root directory (it is the file highlighted in blue in the screenshot below).



Picasso

Add `compile 'com.squareup.picasso:picasso:2.5.2'` to your dependencies block.

Spotify Wrapper

Here, we'll compile a new aar (Android archive library) from the most recent git clone of the Spotify Web API repository on GitHub. See the README for how to compile the source. After you've followed the README, your `app/build.gradle` file should contain the following:

```
// Add this block
repositories {
   flatDir {
        mavenCentral()
        dirs 'libs'
   }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar']) // this line
is already created in Android Studio
    compile 'com.android.support:appcompat-v7:22.2.0' // you may
need this line when targeting earlier Android APIs

    // Add these lines
   compile(name:'spotify-web-api-android-0.1.0', ext:'aar')
    compile 'com.squareup.picasso:picasso:2.5.2'
   compile 'com.squareup.retrofit:retrofit:1.9.0'
   compile 'com.squareup.okhttp:okhttp:2.2.0'
}
```

Note:

- If you are on Windows, you'll need to use `.\gradlew assemble`, not `./gradlew assemble` to build the spotify-web-api-android library.
- These gradle `compile` dependencies are specifically used by the spotify wrapper and you do not need to use them directly. Also don't worry if you don't completely understand these lines. You'll learn more gradle and how it works with Android Studio later in your Nanodegree journey.
- You'll want to build the aar manually with the above instructions. **Do not** download and use the **spotify-web-api-android-0.1.0.jar** from the releases page. The README instructions and the code snippets in this doc are not compatible with the jar in the releases page since that release is outdated.

Build the UI: Search for an Artist, then Return Their Top Tracks

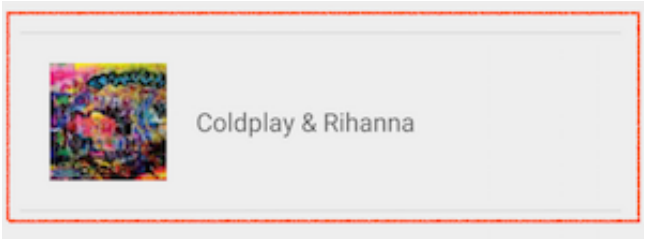
Task 1: UI to Search for an Artist

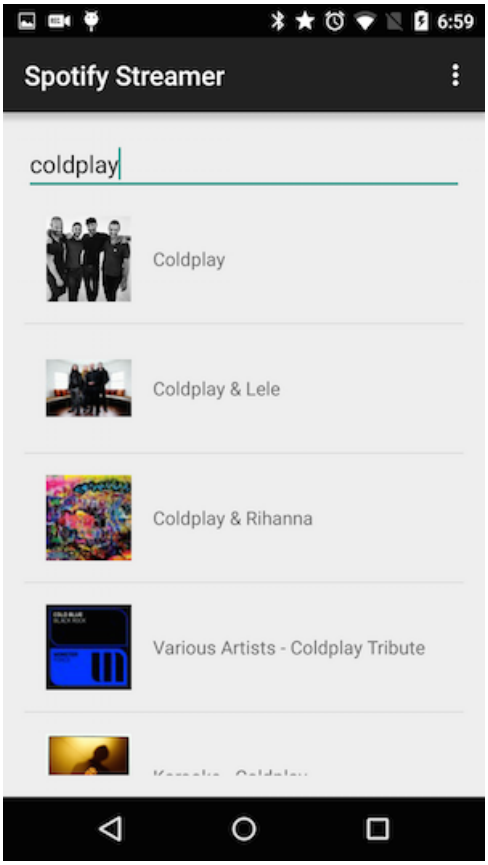
Design and build the layout for an Activity that allows the user to search for an artist and then return the results in a ListView.

Corner case: If no artists are found, display a toast stating this or write a similar message to the layout.

Note: You should design two layouts:

1. For the artist search activity
2. For an individual artist search result





Guidance:

- See Lesson 1 of Developing Android Apps for support on building UI layouts. For extra polish, check out this section of Lesson 5.

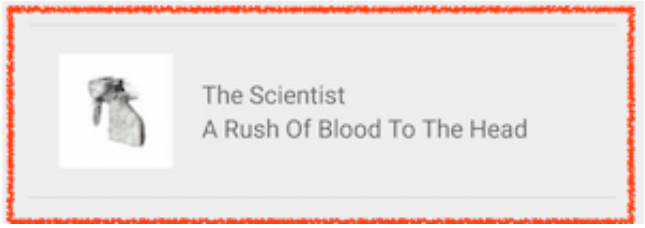
Task 2: UI to Display the top 10 tracks for a selected artist

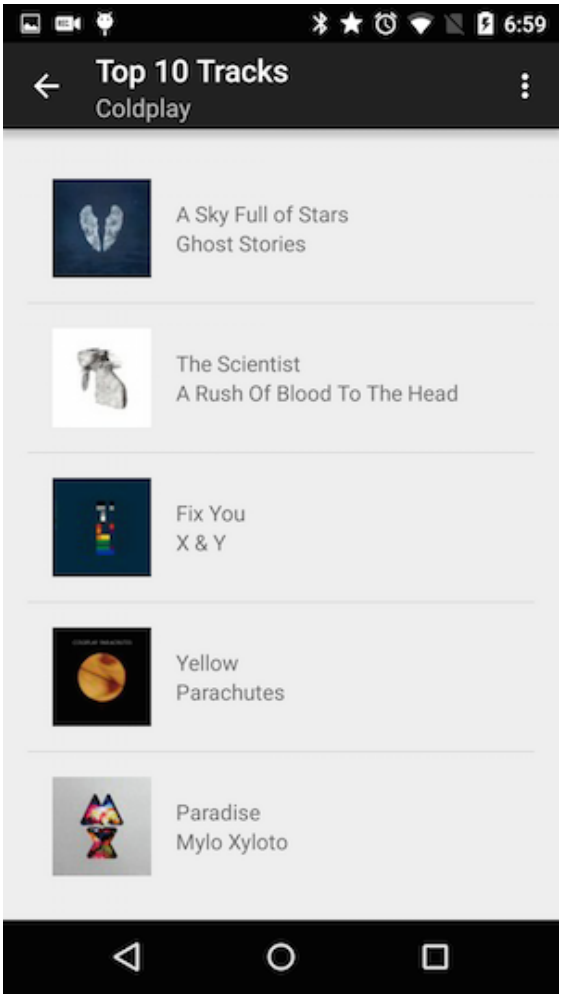
Design and build the layout for an Activity that displays the top tracks for a select artist.

Corner case: If no top tracks are found for the selected artist, display a toast stating this or write a similar message to the layout.

Note: You should design two layouts:

1. For the Activity that will display the tracks in a list
2. For an individual track (thumbnail, album, track)





Task 3: Query the Spotify Web API

Overview

For the two user interface flows described in tasks 1 & 2, you will need to fetch artist and track data from the Spotify Web API to populate your list views. As mentioned earlier, you will use the Spotify Web Api Wrapper to simplify this task since it handles making the HTTP request and deserializing the JSON response for you. You will be able to extract the artist and track metadata you need directly as java objects.

You will issue the following requests to the Spotify Web API:

For task 1, you will need to request artist data via the **Search for an Item** web endpoint.

- Be sure to restrict the search to artists only by including the item type=”artist”.
- For each artist result you should extract the following data:
  - artist name
  - SpotifyId* - This is **required** by the **Get an Artist’s Top Tracks** query which will use afterwards.
  - artist thumbnail image

For task 2, you will need to request track data via the **Get an Artist’s Top Tracks** web endpoint.

- Specify a country code for the search (the API requires this). You can either set a hardcoded String in the query call, or make a preference screen to make the country code user-modifiable.
- For each track result you should extract the following data:
    - track name
    - album name
    - Album art thumbnail (large (640px for Now Playing screen) and small (200px for list items)). If the image size does not exist in the API response, you are free to choose whatever size is available.)
    - preview url* - This is an HTTP url that you use to stream audio. You won't need to use this until Stage 2.

Programming Notes:

- Populate the ListView using a ListView Adapter.
- Fetch data from Spotify in the background using AsyncTask and The Spotify Web API Wrapper

Additional Guidance

Mapping Web API Queries to the Spotify Wrapper Java API

Important Note: Before you write any code using the Web API wrapper, it is important that you familiarize yourself with how the the Spotify Web API works via HTTP.

By doing this exercise, you will not only understand what the wrapper library is doing for you but you'll be able to fully understand the request's parameters as well the JSON response that will be returned.

How do I manually test the Web API via HTTP?

You can do this by submitting test HTTP requests using Spotify's web console or any other REST client of your choice. (@ Udacity, we like clients like PAW or POSTMAN)

Once you're comfortable with the structure of the necessary API requests (and their corresponding responses) in raw form, you can easily implement the same request sequence in Java using the Spotify Wrapper.

Here's a more detailed example:

A search for the artist "Beyonce" is done with this API call (you can make the request via Spotify Web Console's "Search for an Item" page):
https://api.spotify.com/v1/search?q=Beyonce&type=artist

Instead of making this raw request in your app, the Spotify Wrapper allows you to easily make this request without needing to do any JSON parsing:

```
SpotifyApi api = new SpotifyApi();
SpotifyService spotify = api.getService();
ArtistsPager results = spotify.searchArtists("Beyonce");
```

That's it! You can check out the SpotifyService class for the other Spotify API methods that the class maps directly against the other Spotify endpoints. Feel free to dig through the models folder on the Github site to better understand what types of objects these methods return.

Using Picasso To Fetch Images and Load Them Into Views

You can use Picasso to easily load album art thumbnails into your views using:

```
Picasso.with(context).load("http://i.imgur.com/DvpvklR.png").into(imageView);
```

Picasso will handle properly caching the images.

Spotify Streamer, Stage 2: Implementation Guide

Stage 2: Overview

This document should be used as a guide and development plan for Stage 2 of the Spotify Streamer app. It breaks down the build process into smaller, concrete milestones, and provides technical guidance on specific tasks.

Students are expected to have successfully completed Stage 1 prior to beginning Stage 2.

Supporting course material for this project: Developing Android Apps, Lessons 4-6.

Evaluation

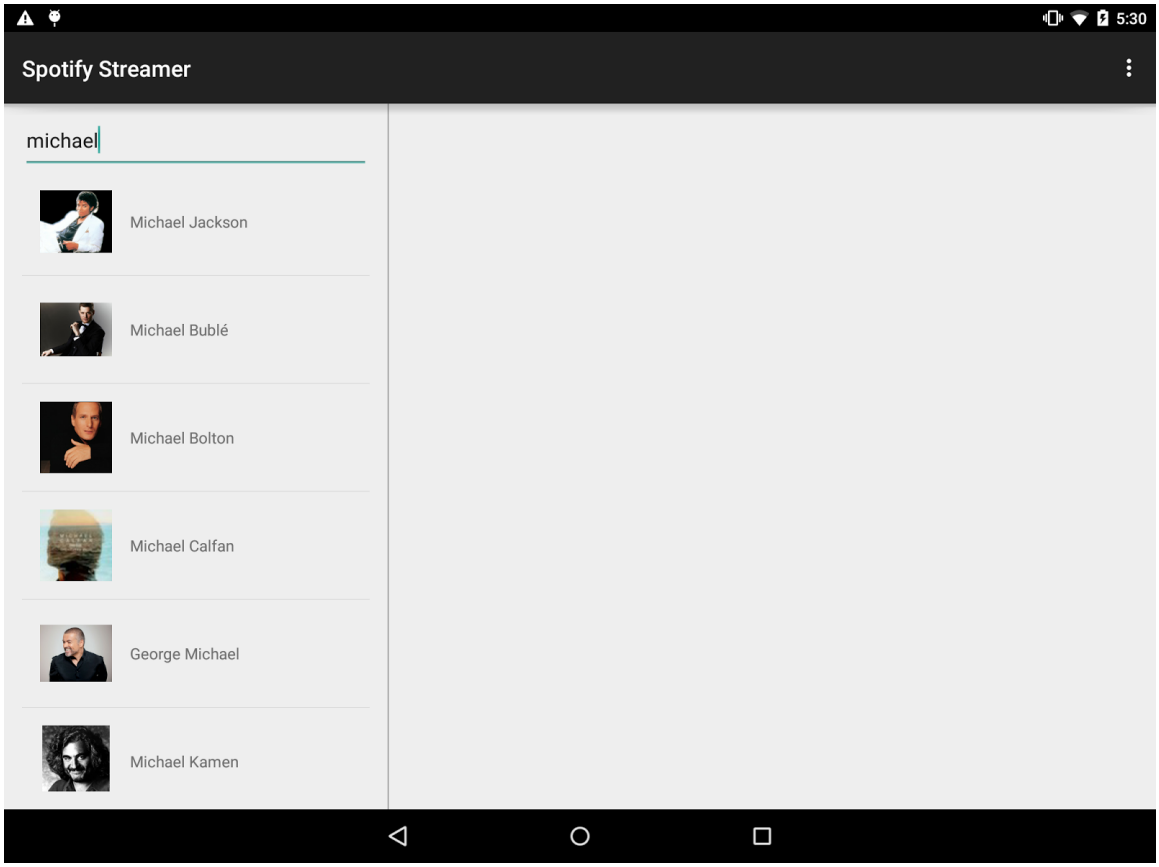
Stage 2 is evaluated against this rubric.

User Experience Design

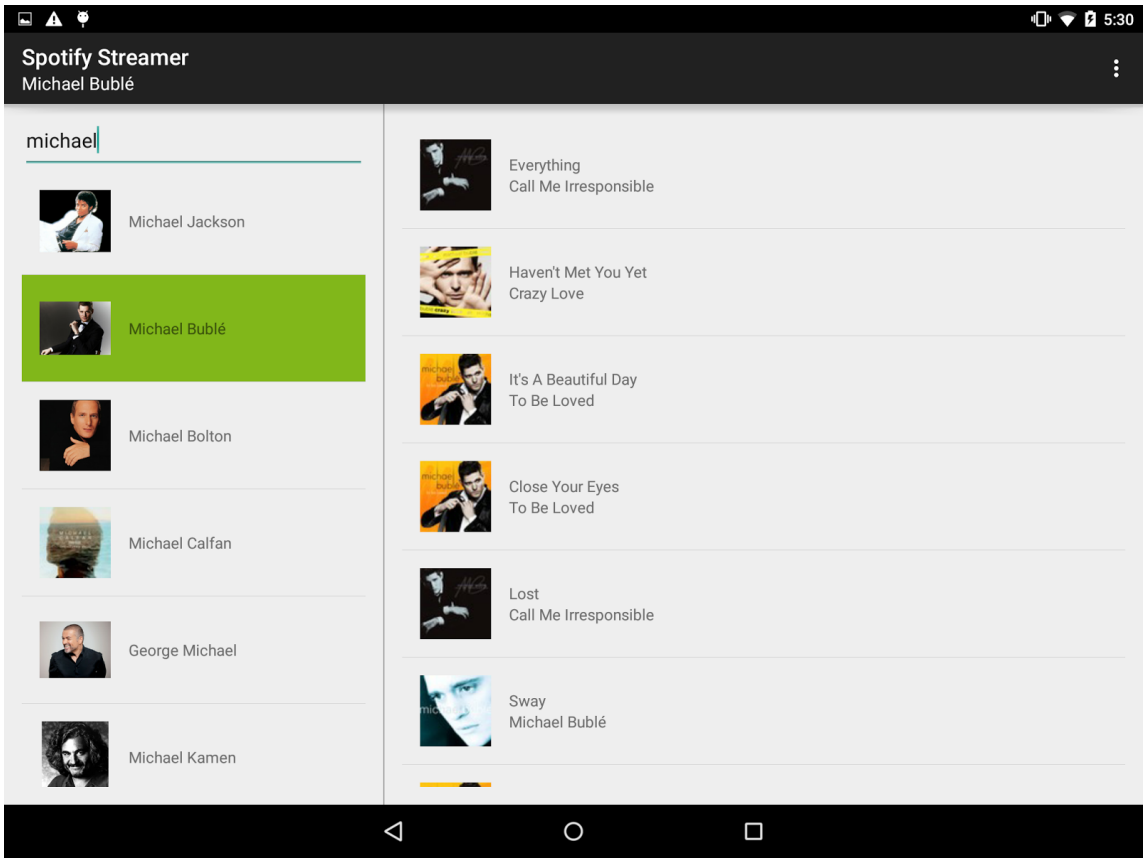
Tablet Interaction Flow

(using a Master Detail Flow)

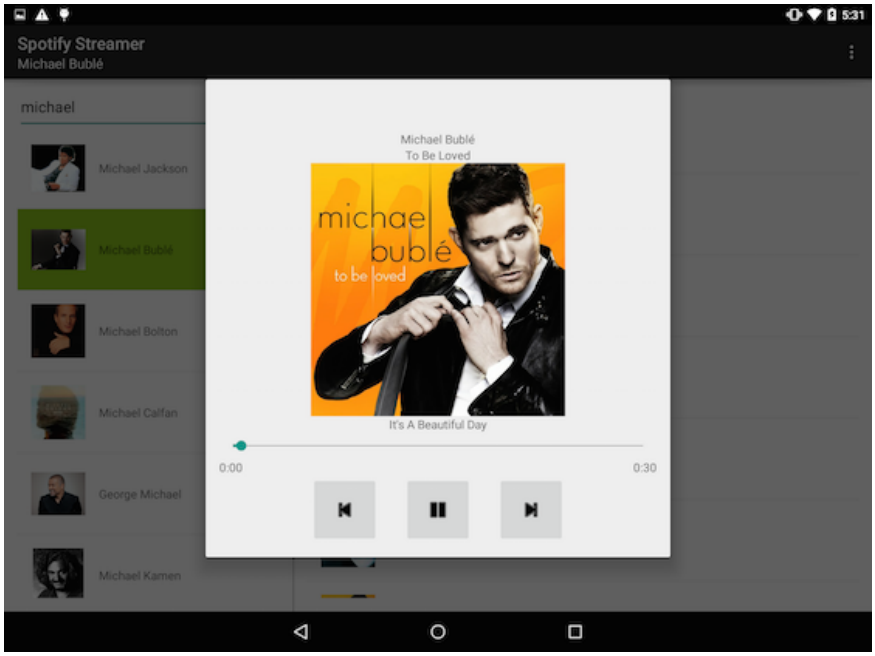
1. Search for an artist.

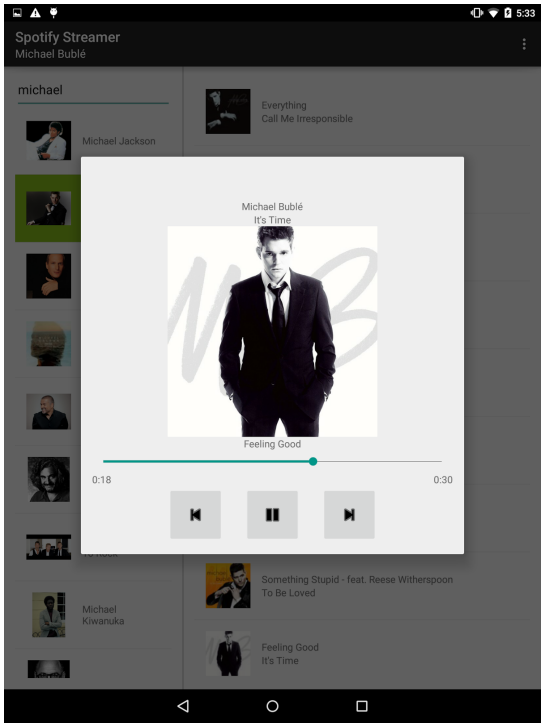


2. Fetch and display the top tracks for the selected artist.



3. Play the selected track preview.

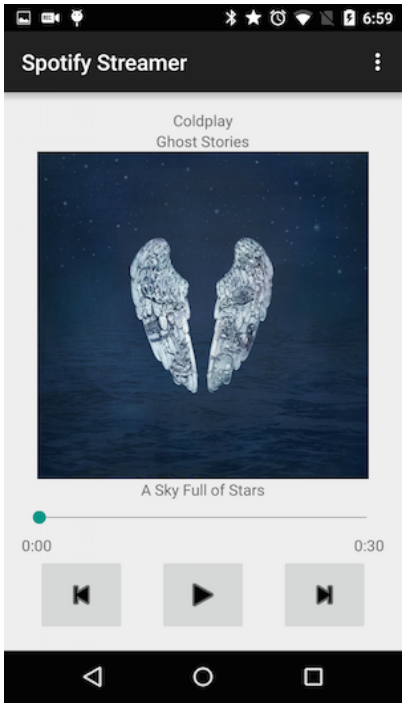




Build a Track Player and Optimize for Tablet

Task 1: Build a Simple Player UI

Use the layout fundamental skills you learned in Developing Android Apps to build a simple track player. You will launch this view via Intent when a user selects a specific track from the Artist Top Tracks view.



This player UI should display the following information:

- artist name
- album name
- album artwork
- track name
- track duration

This player UI should display the following playback controls:

- Play/Pause Button - (Displays “Pause” when a track is currently playing & displays “Play” when a playback is stopped)
- Next Track - advances forward to the next track in the top track list.
- Previous Track - advances to backward to the previous track in the top track list.
- (Scrub Bar) - This shows the current playback position in time and is scrubbable.

To get the icons for the playback controls, you can use the ones that are built-in on Android as drawables. Check out http://androiddrawables.com/ to see all the built-in drawables. The ones used in the mockup are found in the Other section of the site:

- `ic_media_play`
- `ic_media_pause`
- `ic_media_next`
- `ic_media_previous`

Referencing built-in Android drawables involve using the syntax `@android:drawable/{drawable_id}`. For example, `@android:drawable/ic_media_play` refers to the play button drawable.

Task 2: Implement playback for a selected track

You will use Android’s MediaPlayer API to stream the track preview of a currently selected track.

Please consult the guide on using MediaPlayer on developer.android.com

- Remember that you will be **streaming** tracks, not playing local audio files.

- Don’t forget to add the necessary permissions to your app manifest.

Task 3: Optimize the entire end to end experience for a tablet

Migrate the existing UI flow to use a Master-Detail Structure for tablet. If you haven’t done so already, you will want to implement three Fragments for your tablet UI: one for artist search, one for top track results, and another for Playback.

- If you need a review of how to build for tablet, please refer back to Lesson 5 of Developing Android Apps, where the instructors discuss a Master-Detail layout.
- To display the Now Playing screen in a dialog on the tablet and in a normal activity on the phone, you can use a DialogFragment, which can act as a normal fragment (for the phone) or show in a dialog (for the tablet). See the documentation on Dialogs for more information—the section called “Showing a Dialog Fullscreen or as an Embedded Fragment” is particularly helpful.