GitHub Username: asimq74

BandNinja

Description

Music aficionados and nerds alike! Are you looking to read about artists and their albums? Are you looking to find similar artists to the ones you love?

BandNinja gives you instant access to millions of artist and album information.

BandNinja is available on your Android device. Research the right artists and their top albums anywhere, anytime.

With BandNinja, you’ll be able to:

- Access a wealth of artists and their bios

- Access information about their top albums
- Access information about the top tracks of those albums
- View information about artists in your favorite genres
- Search through our lists of Artists
- View customized genres of Artists

Intended User

Anyone who is a music buff and wants quick information about their favorite artists and albums, and would like to discover similar artists.

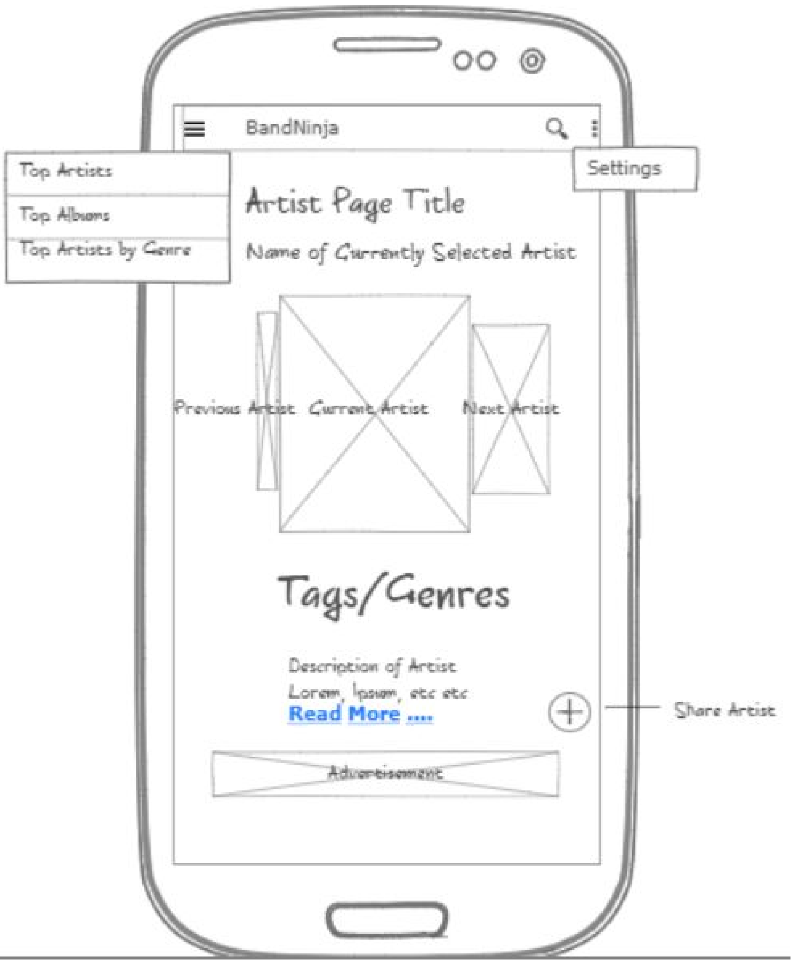
Features

- Saves artist information in a Room database
- Calls Last.fm api service to obtain Artist/Album/Genre data
- Displays Images using Picasso
- Saves Location
- Shows Location-Based Ads using Admob
- Displays a Widget featuring a different artist over a time interval
- Shows album information for each artist
- Shows artist information with photo(s) and descriptions

User Interface Mocks

These can be created by hand (take a photo of your drawings and insert them in this flow), or using a program like Google Drawings, www.ninjamock.com, Paper by 53, Photoshop or Balsamiq.

Main Activity Screen



The Main Activity will be a page that shows a slider adapter with a list of artist images with the current artist in main view and the previous and next artists in the background.

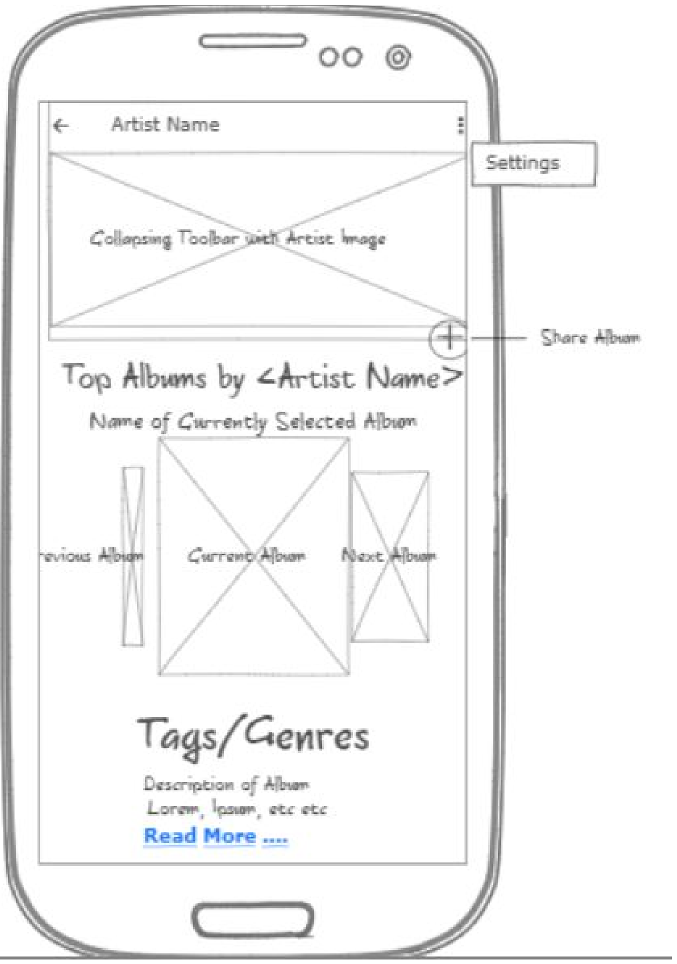
The artist page title above the artist's name will vary based on whether the artists appeared from a specific search result, or otherwise.

Whenever an artist is scrolled into view, the user sees the artist name above the focused image.

Below the focused image, the user will see the genres associated with that particular artist and a shortened summary of the artist below. When the "Read More.." text is clicked, an activity will open with a more detailed view of the summary.

If Google Play Services are enabled, ad banners will appear at the bottom of the page. Search functionality is available via the search icon on the toolbar.

Album Details Screen



The Album Details Activity will be a page that shows a slider adapter with a list of album images with the current album in main view and the previous and next albums in the background.

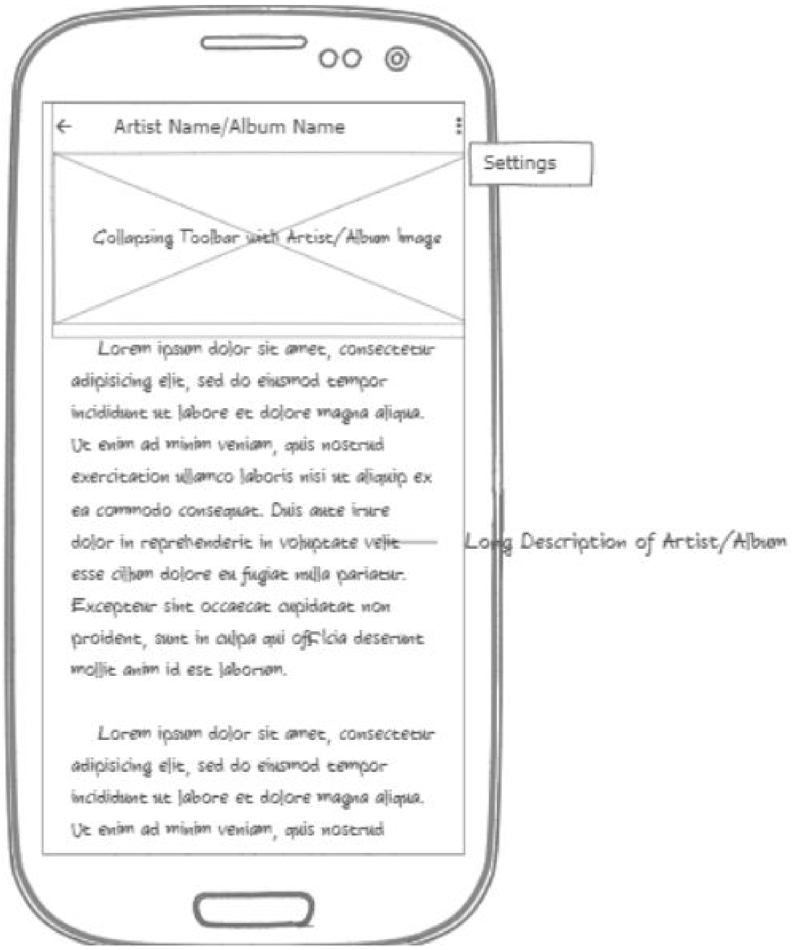
The album page title above the album's name will vary based on whether an artist is associated with the result or otherwise.

Whenever an album is scrolled into view, the user sees the album name above the focused image.

Below the focused image, the user will see the genres associated with that particular album and a shortened summary of the album below. When the "Read More.." text is clicked, an activity will open with a more detailed view of the summary.

There will be a collapsing toolbar above with a back arrow and an image of the artist that appears when scrolled.

Album/Artist Summary Details Screen

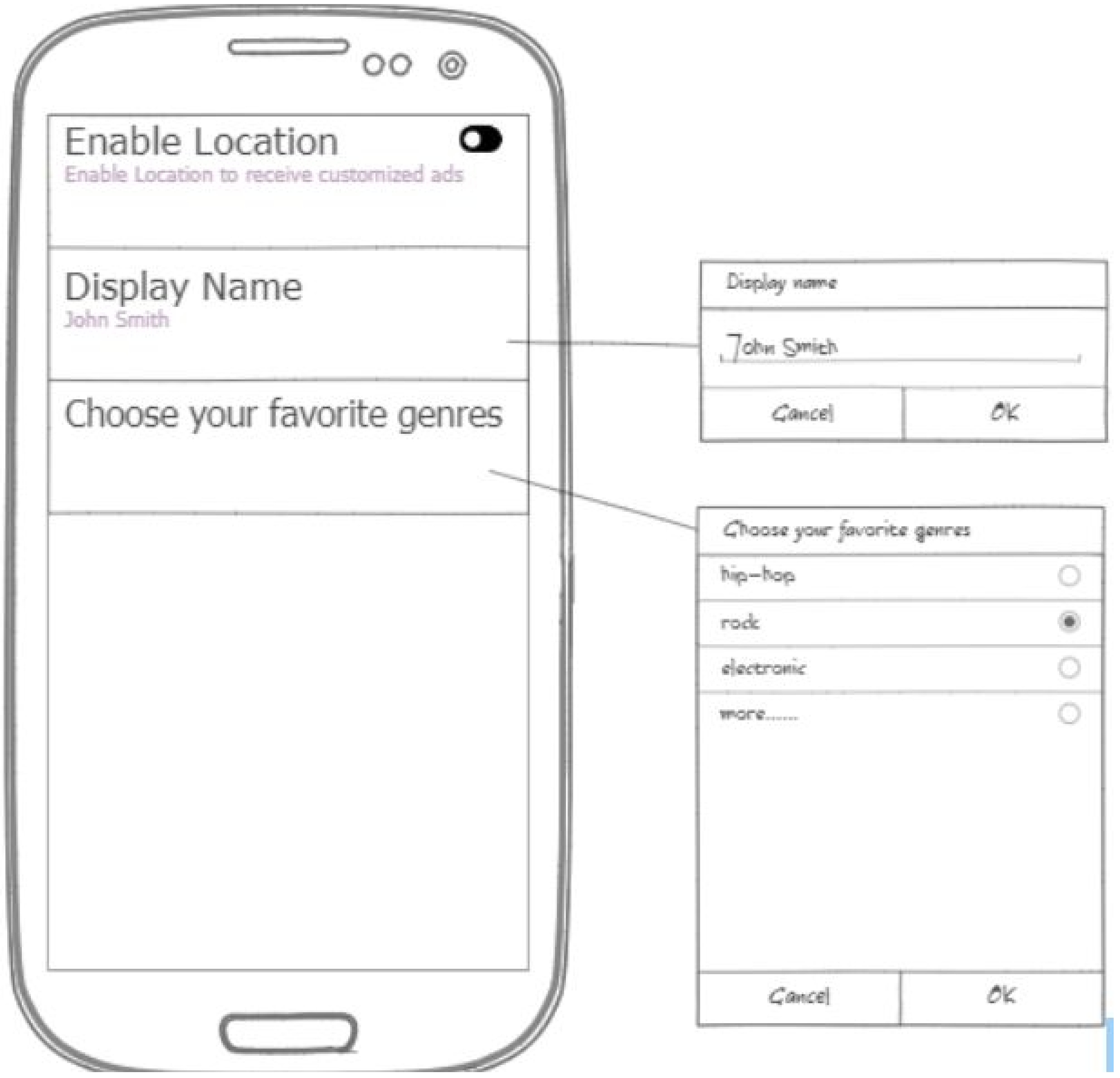


Artist Name/Album Name
Settings
Collapsing Toolbar with Artist/Album Image
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
Long Description of Artist/Album

The Album/Artist Summary Activity will be a page that shows a long description of the summary/article written describing the album in a scrolling and readable format. It is launched when the “Read More..” section is clicked from either the Artists or Album screens.

There will be a collapsing toolbar above with a back arrow and an image of the artist/album that appears when scrolled.

Settings Screen



The Settings screen is invoked from the settings menu and allows the user to customize location services, a display name and to choose your favorite genres. The Enable Location switch will not be disabled if Google Play Services are not available.

Key Considerations

How will your app handle data persistence?

The Room persistence library will be used to save artist, album and genre data. Firebase Job Dispatcher will be used to refresh the data periodically. When an item is viewed (whether from a search result or otherwise), its data is saved to the Room database. This will ensure that server calls are only made as necessary, i.e. if the data does not exist on the local device. This will also enable offline viewing if data is saved to the device.

Describe any edge or corner cases in the UX.

- If Google Play Services are not enabled or there is no internet access, we won't allow the ability to show ads or request location
- If search results are null or empty, display an error message on the screen informing the user of the error.
- If network access isn't available, the app will display any artist/album information that is saved on the device.
- If genres have not been chosen in settings and the user clicks on the "Top Artists by Genre" menu option, the user will be prompted to choose genres in settings
- If genres have been chosen, the user will see menu options for Top Artists by the specific genres they have chosen (up to 5).

Describe any libraries you'll be using and share your reasoning for including them.

Retrofit will be used to make API service calls in order to take advantage of several things including:

- its ability to easily generate the URL in a way that's tied to the specific API call
- to type-safe and make the code interaction with the API more modular
- To take advantage of its callbacks in order to describe error and successful conditions
- Facilitate parsing the JSON into Gson.

```
implementation 'com.squareup.retrofit2:retrofit:2.3.0'
implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
```

Room will be used to store data locally on the device for quick lookup when service calls are not required.

```
implementation 'android.arch.persistence.room:runtime:1.1.1'
annotationProcessor 'android.arch.persistence.room:compiler:1.1.1'
```

Dagger 2 is used for dependency injection, allowing for singletons to be annotated within the code through an application module. This will reduce boilerplate code and lines of code in general.

```
implementation 'com.google.dagger:dagger:2.15'
annotationProcessor 'com.google.dagger:dagger-compiler:2.15'
```

Picasso will be used to load images provided by urls from the API calls

```
implementation 'com.squareup.picasso:picasso:2.5.2'
```

Butterknife will be used to bind data to UI widgets in most cases, eliminating the need to programmatically define and bind them.

```
implementation 'com.jakewharton:butterknife:8.8.1'
annotationProcessor 'com.jakewharton:butterknife:8.8.1'
compileOnly 'com.jakewharton:butterknife-compiler:8.8.1'
annotationProcessor 'com.jakewharton:butterknife-compiler:8.8.1'
```

Android Architecture/Lifecycle Components will be used to establish view models for the various activities and fragments, so they can seamlessly respond to changes in data that affect the UI.

```
implementation 'android.arch.lifecycle:livedata:1.1.1'
implementation 'android.arch.lifecycle:extensions:1.1.1'
implementation 'android.arch.lifecycle:viewmodel:1.1.1'
```

Cardslider is a custom material design RecyclerView LayoutManager that will allow the user to swipe through cards with images while being able to view the next few images in the background.

```
implementation 'com.ramotion.cardslider:card-slider:0.3.0'
```

Google Location through the Play Services API is used to determine location if the user allows and to provide that location to display customized ads.

```
implementation 'com.google.android.gms:play-services-location:16.0.0'
```

Google Mobile Ads through the AdMobi Play Services API is used to display mobile ads (test key only) and can display customized ads if the user provides their location.

```
implementation 'com.google.android.gms:play-services-ads:17.1.2'
```

Firebase Job Dispatcher is used to schedule a background task to refresh the user stored data with data from service calls on a periodic basis

```
implementation 'com.firebase:firebase-jobdispatcher:0.8.5'
```

Describe how you will implement Google Play Services or other external services.

Google Mobile Ads will be used to display mobile ads on the UI of the main screen.

Google Location services will be used to determine location and allow the Mobile Ads to consume the location in order to provide customized ads for that location.

Firebase Job Dispatcher will be used to refresh user data from the server data periodically.

Next Steps: Required Tasks

This is the section where you can take the main features of your app (declared above) and break them down into tangible technical tasks that you can complete one at a time until you have a finished app.

Task 1: Project Setup

- Obtain Last.fm API key and store it in the project properties file so the application has access to the key (Note: do not display the key in any public GitHub repositories)
- Create a project in Android Studio called BandNinja and add it to a GitHub repository.
- Configure the following libraries (described above) in the application's build.gradle file:
  - Retrofit
  - Room
  - Dagger 2
  - Picasso
  - Butterknife
  - Android Architecture/Lifecycle Components
  - Cardslider
  - Google Location through the Play Services API
  - Google Mobile Ads through the AdMobi Play Services API
  - Firebase Job Dispatcher
- Add a buildConfigField and resValue declarations for the Last.fm API key in the application's build.gradle file

If it helps, imagine you are describing these tasks to a friend who wants to follow along and build this app with you.

Task 2: Implement UI for Each Activity and Fragment

- Build UI for Main Activity that will allow the user to search by artist and will display the appropriate search icons on the toolbar along with ads.
- Build UI for Album Details Activity that displays the album details by artist
- Build UI for Settings Activity that allows the user to change application settings
- Build UI for Article Details Activity that shows the full artist/album details
- Create widget that allows an artist search from the device's main screen
- Create a fragment with a recycler view and adapter that displays artists/albums and related information. This fragment can be used with both the Main activity and Album Details Activity to display information
- Implement menu options for the Main Activity

Task 3: Implement Retrofit Service Calls

- Create Retrofit Client Instance in order to perform api service calls
- Create Retrofit API interfaces with annotated methods for each service call that will be made (specifying URL and parameters)
- Implement service calls and handle results appropriately

Task 4: Implement Room Database

- Create and configure Room Database Instance
- Create entity pojos for each of the data objects (Artist, Album, Tags, Tracks, etc)
- Create interfaces for all the DAOs

Task 5: Implement View Models/LiveData

- Create LiveData objects that are accessed through the ViewModel that will be used by the UI to detect changes. These will mostly be driven by results from api calls and accessing data from the Room database
- Implement ViewModels that contain methods that allow the UI to access LiveData objects and observe them

Task 6: Implement Location and Ad Google Play Services

- Implement Google Mobile Ads will be used to display mobile ads
- Implement code to allow user to initiate Location Services from the settings screen
- Allow the Google Mobile Ads to be built using the location if location services have been enabled.
- Firebase Job Dispatcher will be used to refresh user data from the server data periodically.

Task 7: Implement Mechanism to Synchronize Data with the Server

- Implement Firebase JobDispatcher to periodically run code that refreshes application saved data from the api service

---

Submission Instructions

- After you’ve completed all the sections, download this document as a PDF [File → Download as PDF]
  - Make sure the PDF is named “**Capstone_Stage1.pdf**”
- Submit the PDF as a zip or in a GitHub project repo using the project submission portal

If using GitHub:

- Create a new GitHub repo for the capstone. Name it “**Capstone Project**”
- Add this document to your repo. Make sure it’s named “**Capstone_Stage1.pdf**”