
- Description
- Intended User
- Features
- User Interface Mocks
  - Screen 1
  - Screen 2
  - Screen 3
  - Screen 4
  - Screen 5
- Key Considerations
    - How will your app handle data persistence?
    - Describe any corner cases in the UX.
    - Describe any libraries you'll be using and share your reasoning for including them.
    - Describe how you will implement Google Play Services.
- Next Steps: Required Tasks
  - Task 1: Project Setup
  - Task 2: Implement UI for Each Activity and Fragment
  - Task 3: Create Room Database
  - Task 4: Download Initial Data
  - Task 5: Create PlayerService
  - Task 6: Create a SettingsFragment
  - Task 7: Implement Admob

GitHub Username: langhimebaugh

Dead Streams

Description

Stream Grateful Dead music concert/shows from the Internet Archive (archive.org). Audio is from soundboard recordings or audience uploads and free to use.

Intended User

This App is intended for Grateful Dead music fans.

Features

Main features of the app:

- Browse concerts/shows.
- Search for songs.
- Filter concerts/shows by year.
- Sort by popularity.
- Save concerts/shows as favorites for easy playback.
- Create a playlist with choice of songs.
- Random Play of songs.
- Widget with Play/Pause control.
- Settings to be determined.

Additional notes:

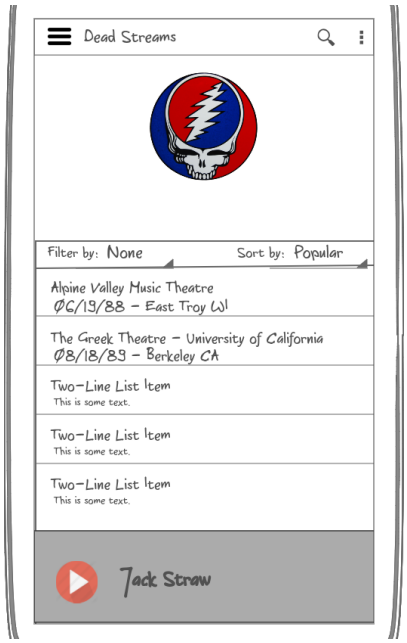
- App will conform to common standards found in the Android Nanodegree General Project Guidelines
- App will be written solely in the Java Programming Language
- App will utilize stable release versions of all libraries, Gradle, and Android Studio.
- App will integrates a third-party library.
- App will validate all input from servers and users. If data does not exist or is in the wrong format, the app will log this fact and not crash.
- App will include support for accessibility. This includes content descriptions, navigation using a D-pad, and, if applicable, non-audio versions of audio cues.
- App will keep all strings in a strings.xml file and enable RTL layout switching on all layouts.
- App will provide a widget to provide relevant information to the user on the home screen (play & pause of music streams).
- App will integrate two or more Google services. Google service integrations can be a part of Google Play Services or Firebase. I will use Firebase Storage and either Admob or Analytics.
- Each service imported in the build.gradle will be used in the app.
- App theme will extend AppCompat.
- App will use an app bar and associated toolbars.
- App will use standard and simple transitions between activities.
- I will download and manipulate some data from Archive.org and maybe combine with data from another source, then place into a few JSON files on Firebase Cloud Storage. The JSON files will be downloaded the first time the app is run. Data will be placed into a local ROOM database.
- App stores data locally by implementing Room and LiveData and ViewModel are used when required and no unnecessary calls to the database are made.
- If Content provider is used, the app uses a Loader to move its data to its views.

- Will implement at least one of the three:
  If it regularly pulls or sends data to/from a web service or API, app updates data in its cache at regular intervals using a SyncAdapter or JobDispatcher.
  OR
  If it needs to pull or send data to/from a web service or API only once, or on a per request basis (such as a search application), app uses an IntentService to do so. (I think I will use IntentService when downloading JSON from Firebase Cloud Storage.)
  OR
  If it performs short duration, on-demand requests(such as search), app uses an AsyncTask.
- If Admob is used, the app displays test ads. If Admob was not used, student meets specifications.
- If Analytics is used, the app creates only one analytics instance. If Analytics was not used, student meets specifications.
- Location will not be used, student meets specifications.
- Maps will not be used, student meets specifications.
- Identity will not be used, student meets specifications.

==========

User Interface Mocks

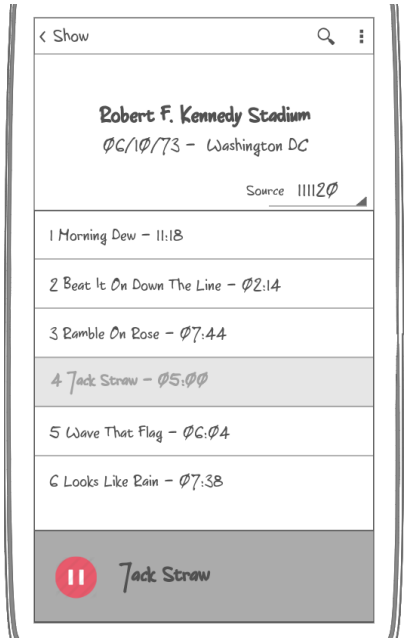
Screen 1



Initial/Main screen will display concerts/shows sorted by popularity with option to change sort criteria. Option to filter by year. Search option on each screen. Play controls at the bottom will

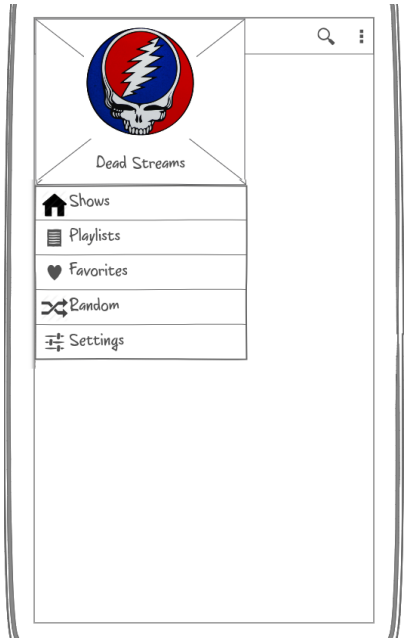
default to users last music selection and change when clicking a list item. Initial use will hide play controls. Plan to use image without copyright restrictions.

Screen 2



Detail screen will list songs from the show or the playlist selected. Will need to click on individual song to change from currently playing/selected show to the show displayed in the detailed screen. This allows browsing other shows, while the current selection is not interrupted. May provide an additional play button near the top to initiate the selection and playing of the show/playlist on this detail screen. Favorite button to add show to favorites (not shown in drawing) will be up by the show name. Playlist button (next to song name in the list item & not shown in drawing) OR possibly long pressing list item to add song to a playlist. Play controls at bottom of screen.

Screen 3

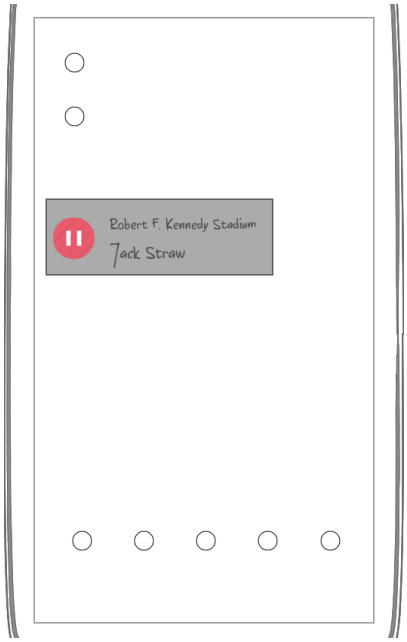


Menu screen will allow user to change screens between Shows, Favorites, Playlists, Random Play, and settings.

The favorite screen will resemble Screen 1, as it is a limited selection of shows. Clicking the list item will display Screen 2.
The playlists screen will resemble Screen 1 but displaying playlist names and have the ability to add/delete playlists. Clicking the list item will display a screen similar to Screen 2 with playlist title displayed instead of the show info at the top.

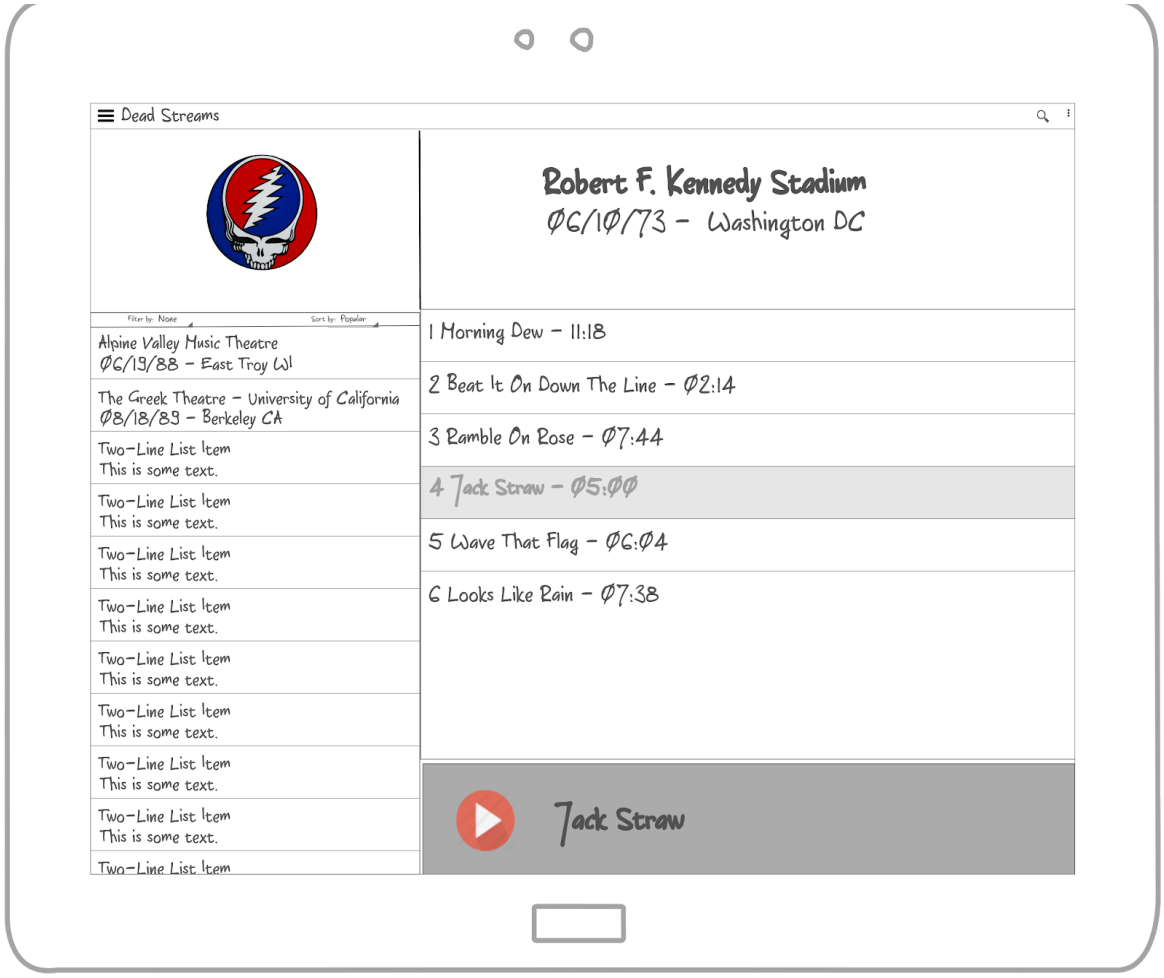
MainActivity, ShowListFragment, PlayListFragment, FavoritesFragment, RandomPlayFragment, SettingsFragment, PlayBackControlsFragment, PlayerService, DetailActivity, DetailFragment

Screen 4



App Widget will allow user to control Audio Playback.

Screen 5



Tablet Layout

Key Considerations

How will your app handle data persistence?

JSON files will be downloaded from Firebase Cloud Storage the first time the app is run. The data will be placed into a local ROOM database. A Content Provider might be used for the widget. onSaveInstanceState will be used to save last song playing to be used upon startup and on rotation.

Describe any edge or corner cases in the UX.

The player will appear on the bottom of every screen. If the user hits the back button or opens a different app, they can see what's currently playing either from viewing the widget or the notification and control playback there. They can also click either one to return to the app and view the currently playing song from there.

A user message will be displayed if no internet connection exists, as playback will not be available.

Detect loss of audio focus (ie incoming phone call), reduce the volume or stop playback depending on which is appropriate.

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

Libraries may include but are not limited to:

- Android Support for recyclerview, design, etc
- Architecture Components (Room, etc)
- Exoplayer for MP3 audio stream playback
- DataBinding for binding UI components in layout
- Firebase Storage for my JSON files
- OkHttp for faster retrieval of JSON files via HTTP client
- GSON for JSON serialization
- Picasso for image loading and caching

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

Firebase Storage for my JSON files

implementation 'com.google.firebase:firebase-storage:16.0.1'

Follow instructions @ https://firebase.google.com/docs/storage/android/start?authuser=0

Admob for advertising upon initial app startup

implementation 'com.google.android.gms:play-services-ads:15.0.0'

Follow instructions @ https://developers.google.com/admob/android/quick-start

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

- Create a new Android Studio project
- Configure libraries
- Add INTERNET, ACCESS_NETWORK_STATE and other permissions to the manifest.

Task 2: Implement UI for Each Activity and Fragment

- Build UI for MainActivity
- Build UI for ShowListFragment, PlayListFragment, FavoritesFragment, RandomPlayFragment, SettingsFragment
- Build UI for PlayBackControlsFragment
- Build UI for DetailActivity, DetailFragment

Task 3: Create Room Database

- Create Tables: Shows, Sources, Songs, Playlist
- Favorite Shows will be identified by an additional field in the Shows table.
- SongID will be added to the Playlist table. Maybe ID, Name & array of SongID.

Task 4: Download initial data

- Use Intent Service to download JSON from Firebase Cloud Storage.
- Populate the database

Task 5: Create PlayerService

- Implement SimpleExoPlayer as a service.
- Provide audio playback and create notifications that allow playback control.
- See ExoPlayer documentation.
- See ExoPlayer's Demo application https://google.github.io/ExoPlayer/demo-application.html.

Task 6: Create a SettingsFragment

- Handle Preferences. To be determined.

Task 7: Implement Admob

- Display interstitial ad on app startup.
- Will follow instructions here...
  https://developers.google.com/admob/android/interstitialreate layout

---

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**"