How to Use this Template

1. Make a copy [File → Make a copy...]
2. Rename this file: "**Capstone_Stage1**"
3. Replace the text in green

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"

UPDATE AS OF JUNE 30, 2016:

Due to time limitations in completing Capstone 2 by my July 14 deadline,
I am removing the "download" functionality (Task 9) from the 1st version of this app.
And because Task 10, a data synchronization service related to offline listening, is entirely dependent on Task 9, both Tasks 9 and 10 are removed from the Capstone 2 deliverable.
And because much of Task 8 relates to download, the Settings Task 8 is now modified to Implement only *Font Size and Selection* settings.

UPDATE AS OF JULY 25, 2016:

I am delaying the Voice Search feature (Task 13) until next release.

Added a companion Wear app that functions to toggle on/off main app playback.

Decided to submit the current version to Udacity.
This effort represents ~8 weeks of continuous app development.

Table of Contents

GitHub Username: LeeHounshell

Radio Mystery Theater

Description

This app lets you listen to 1399 one-hour radio broadcasts of the “CBS Radio Mystery Theater.” This app provides an easy, quality listening experience. App features continuous, hands free playback where possible. Supports background playback. The app properly handles incoming calls. Supports voice search. ~~Download episodes for offline listening or~~ stream shows live. All radio shows are property of CBS Corporation and are publicly available to download free from http://cbsrmt.com

Two versions: The free app gives each *authenticated* user access to any 10 shows at no cost. And in-app purchases grant access to the remaining shows; and remove advertising. The paid version is entirely unlocked, with no advertising. Both versions are full-featured.

App Intended User

“Commuters”, “Seniors” and “Disabled” are the primary intended users. This app is designed to be *easy to see and easy to use*. It provides access to radio dramas with minimal device interaction, making “Radio Mystery Theater” ideal for listening while on the go. Easy use lets Seniors and Disabled play too. No internationalization is provided; all shows are in English only.

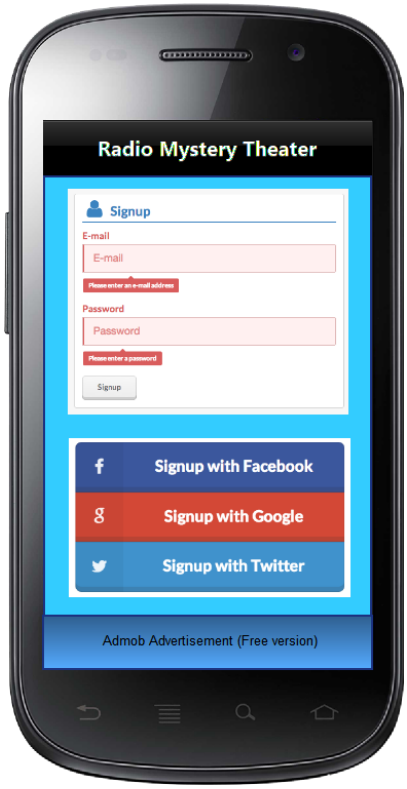
Features

- One button play/pause/resume control with background operation
- Automatically resumes playback on app startup
- Continuous playback skips to next unheard and available show
- Playback Slider Control for manual re-positioning
- Stream shows live, or do a (resumable) download for offline listening
- Voice Search
- In-App Purchases
- SQLite
- Google Analytics
- Admob
- Firebase Integration
- Material Design Theme extends from AppCompat
- CoordinatorLayout + AppBarLayout + CollapsingToolbar
- Butterknife
- A Gradle build 'preprocessor' which permits #IFDEF 'flavor' and #IFDEF 'buildType' in Android Java code

User Interface Mocks

These mockups were created with *Pencil*. The Pencil app has limitations, but is free to use. These mockups should be considered “guidelines” and not exact representations of the various possible screen states. Displayed colors do not represent final color selection, which will be from the Material Design palette. App styling will change to better support Material Design.

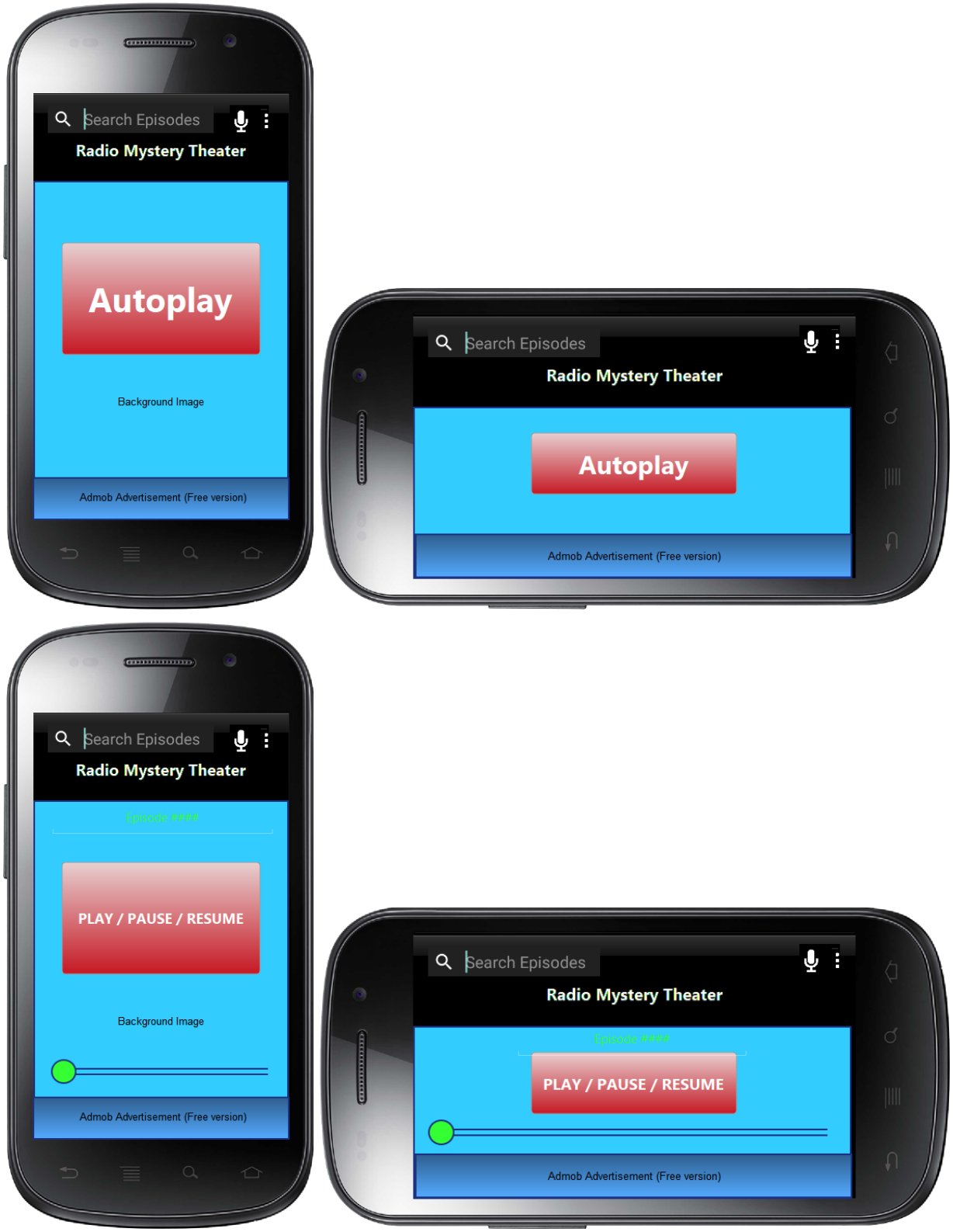
Screen 1 - The AuthenticationActivity - Tablet and Phone



This app requires Firebase User Authentication to use.
See https://github.com/firebase/firebase-simple-login-java/tree/master/docs/v1

Authentication is an app installation event. After new user authentication, a JSON download of the Radio Mystery Theater *database* commences from the Firebase DB master. A progress dialog displays as episode information for 1399 shows is downloaded and extracted from JSON and written via Content Provider to the local SQLite DB. The local SQLite database contains information about: Episodes, Actors, Writers and the authenticated user’s Configuration. Once this information is loaded, the app automatically continues to *Screen 2*, below.

Screen Set 2 - The AutoplayActivity - Tablet and Phone



After authentication, the AutoplayActivity (shown above, top) appears. There is a large central button labeled “Autoplay.” The above 4 screens generate from 2 layout XML files (port and land); and the progress-bar remains *invisible* until Autoplay is pressed. The Autoplay button has 3 states: (Auto)*Play*, *Pause* and *Resume*. Once pressed, playback of the next available episode starts; the big red button label changes to “*Pause*”; and the progress-bar appears. If pressed again then audio playback pauses and the button label changes to “*Resume*.” When the episode finishes playing, it is marked in the database as “HEARD” and the next show automatically loads and begins playback. The button text now says “*Play*” until the transition completes and playback of the newly loaded episode begins, at which time the red button label changes to say “*Pause*” again. If left unchecked, this process repeats in a loop - until all 1399 Radio Mystery Theater episodes have finished playing and are marked in the database as “HEARD.”

There is some intelligence in the automatic playback. Auto playback is intended for the next show episode number, in sequence, unless:

- The show has already been heard, in which case auto playback skips to the *next* show.
- The user is offline. Then playback begins with the first downloaded *and* unheard show.
- There are no available unheard shows. In that event, a dialog appears. The dialog says that the “*internet connection is down*” or that “*all episodes have been heard*” as may be.

The Firebase database master gets updated from the local SQLite DB by a SyncService. Each SQLite row has a “dirty” flag component that indicates if that row’s recent changes also exists in the online Firebase database. Whenever the app regains network connectivity from a period of being offline, the SyncService wakes up and scans the local SQLite data for any *dirty* records. Any found records are uploaded to Firebase. The *dirty* flag is turned off after a successful sync.

The *positioning button* for the audio playback progress-bar can be manually adjusted at any time. It always shows the current playback position, and auto advances as playback progresses.

If an episode is playing and a long-press on the big red button happens, then playback skips backward 30 seconds. Playback begins again from the new position automatically (regardless of prior playback state). This lets a user easily go back and re-listen to just-heard-it content.

All played episodes are logged to Google Analytics and to the user’s Firebase history.

A big red button supports the simple use-case of an “on/off” switch.

The *AutoplayActivity* becomes the main app startup screen once Authentication has completed.

Screen Set 3 - Search Results - Phone

The user can Search Episodes using speech and/or text from *AutoplayActivity*. The above 2 screens show the results of a search. In the header above, "####" is replaced by the number of matches and "<query>" is replaced with the user's actual query text. Each query is logged to Google Analytics and the user's Firebase history. So is any selected result from the search. Note the *colors* of returned list-items will change, depending on if they are downloaded or have been heard. *Downloaded episodes are marked by Green colored Title and Description text. Heard episodes are marked with a darker grey colored background for the list entry.*

Search is context-aware in that an effort is made to place queries into the frame of "finding shows about Radio Mystery Theater." An effort is made to understand limited basic English grammar by discovering key phrases and then executing them as equivalent SQL. If the user says or enters "Help," a browser Intent opens to: http://harlie.com/radiotheater/help.html
Some examples of planned search phrases:

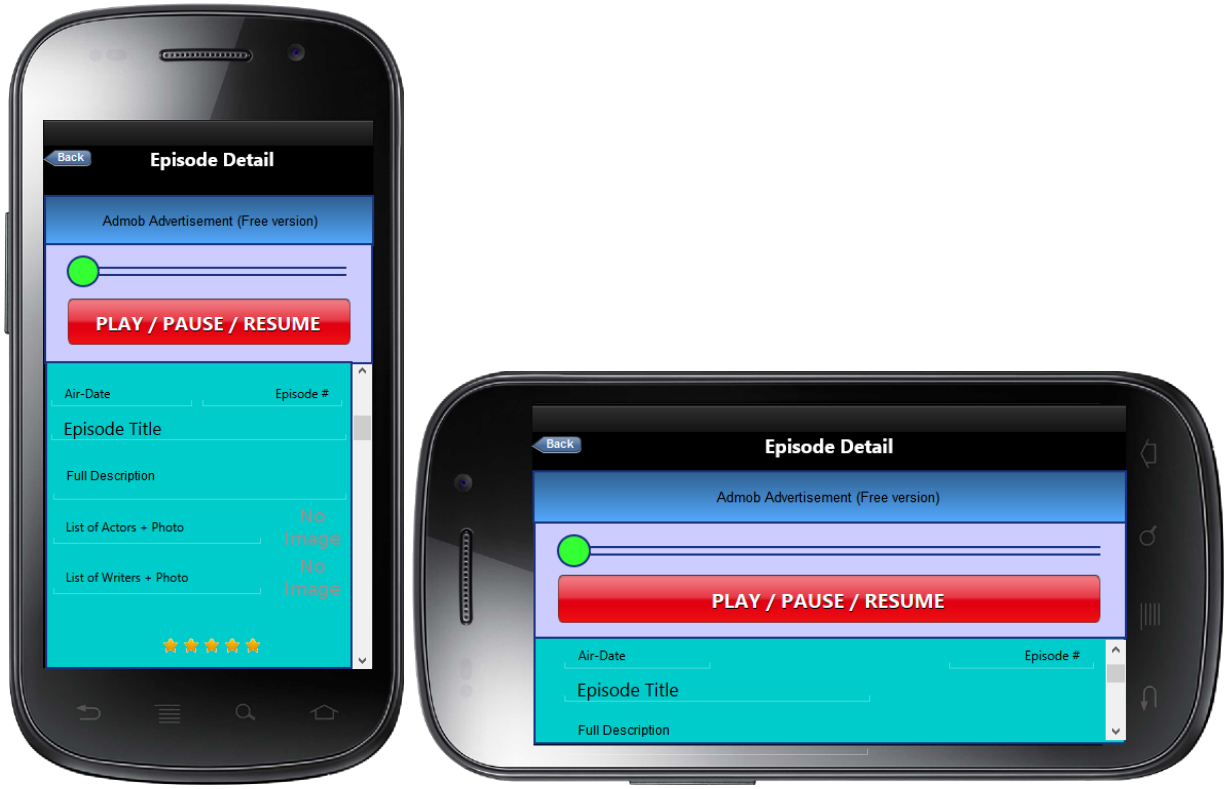
Episode [910]. -or- just [*910].*	start playing the named episode number
Show [all | offline | unheard] episodes.	matching episodes by some qualifier
Episodes with [5] star ratings.	matching show ratings
[Unheard] shows from [Month] [Year].	matching show air dates
By writer "Dan" [with actor "Fred"].	matching writer(s) and/or actor(s)
About ["time travel"].	matching part of a Description or Title

Search phrases may be combined to form interesting queries. E.g.

> *Unheard shows from January 1976 by writer "Dan" with actor "Fred."*
> *Shows I have not already heard from 1976 about "time travel."*

If the Search Results contain only a single list item, then the app *directly displays the match* by transitioning to *Screen 4*, below and auto starting playback. Any list item that is clicked also transitions to Screen 4, but playback is not auto initiated in that case.

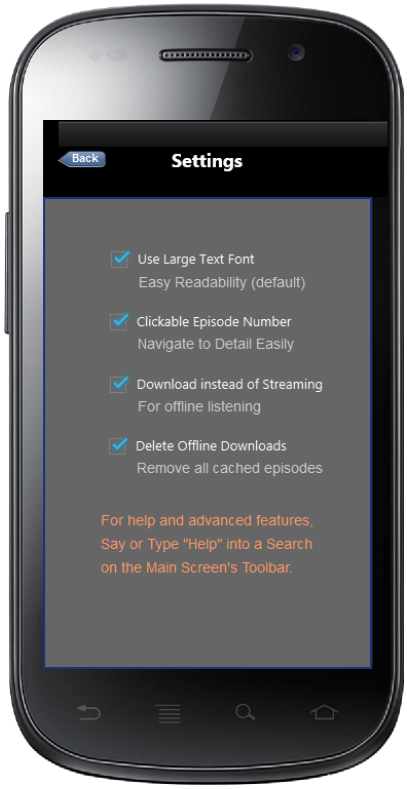
Screen Set 4 - Episode Detail - Phone



The above 2 screens display episode detail. They are reached from a Search query *result selection*, or if the AutoplayActivity's "*Episode #*" is set to *clickable* and the user clicked on it. See the Settings Screen below.

If playback commences to the next episode while viewing an EpisodeDetail, then the detail view will auto update to reflect the now-playing episode.

Screen 5 - Settings - Tablet and Phone (Layout must change. See Task 9)



The above Settings screen allows for:

- Reducing text size. (default is large text)
- Enabling a clickable "*Episode #*" for quick access to Episode Detail. (default is off)
- Toggling between Download and Streaming access. (default is streaming)
- Deleting the cache of all "downloaded episodes." (initiate delete by checking this box)
- Guiding the user to invoke "Help" from "Search Episodes.."

Settings are stored in Global Shared Preferences for easy access.

Screen Set 6 - Combined Search Results and Episode Detail - Tablet

These are combined master-detail views for a Tablet display. Reference the sections for *Screen Set 3* and *Screen Set 4* to find additional detail about the information displayed here. Prior to a list selection being made, the *EpisodeDetailFragment* defaults to showing the first unheard and available episode. If the user arrives at the EpisodeDetail view from a *clickable* "*Episode #*" then the implicit search query used to populate the *EpisodeListActivity* is "*All unheard.*"

If playback commences to the next episode while viewing an EpisodeDetail, then the detail view will auto update to reflect the now-playing episode.

Key Considerations

How will your app handle data persistence?

A local SQLite database (accessed via Content Provider) manages the show ***episodes***, ***actors***, ***writers*** and a user's authenticated Firebase ***configuration***. A Firebase backend is the source for this data. The Firebase Android API will access and update all Firebase content. Firebase manages data, user authentication, show access permissions and analytics. The local SQLite database initializes from Firebase content when the Radio Mystery Theater app is first run. And a local app cache directory contains copies of any downloaded shows, 1 per mp3 file.

Describe any corner cases in the UX.

If a phone call is received while a show is playing, the show pauses until the call completes.
If a voice search is made while a show is playing, the show pauses until the search completes.
If the search results in a single matched episode, then autoplay launches that show.
If a show is playing and a new show is selected to play, then playback of the original show halts.

Playback must honor the Settings value for "*Download instead of Streaming*." In the Download case, the episode is downloaded to a local cache directory while a progress bar displays. Once the download completes, playback automatically starts. Interruption of a download will be handled by the Google *resumable media download* functionality.

The *RadioTheaterService* and/or *SyncService* may already be running. Ensure that they are started once, properly, and are destroyed when the AutoplayActivity calls *onDestroy*.

A race condition might exist relating to the *dirty* flag used by SyncService. Try to avoid it, but worst-case is that Firebase master will miss a history update.

When displaying the tablet combined EpisodeList and EpisodeDetail view **prior** to a selection being made, the *EpisodeDetailFragment* defaults to showing the first unheard and available episode. And if the user arrives at the EpisodeDetail view from a *clickable* "*Episode #*" then the implicit search query used to populate the *EpisodeListActivity* is "*All unheard.*"

Libraries

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

I plan to evolve this app for Android TV, Wear and Auto. For that reason this app will use the new Android Media Libraries; specifically the MediaBrowserService API and MediaSession callbacks. When possible (during Autoplay, for example) media selection and playback will be controlled by the Radio Mystery Theater app via automatic browsing and selection of media.

When a network connection is required to load JSON and/or media, the user sees a circular progress dialog. The "Circle Progress View" library will create that view when loading the local database for the first time, and when an episode is downloaded. Also, "Circular Seekbar" by RaghavSood@appaholics.in will create a playback adjustment control. Add the following to build.gradle for the Circle Progress View:

```
compile 'com.github.jakob-grabner:Circle-Progress-View:v1.2.2'
```

Additionally, these more common libraries will also be used:

The Google API Client Library supports *resumable download* of episode media.
https://developers.google.com/api-client-library/java/google-api-java-client/media-download#implementation_details

The Google Translate Services Library provides voice to text translation.
The Google Play Ad Services Library provides AdMob integration.
The Google/Firebase Analytics Libraries provide Analytics integration.
The Firebase Services Libraries provide authentication, tracking and data services.
The SQLite Library provides local database integration.
The Android AppCompat Libraries provide Android backward compatibility.
The Material Design Support Library allows for Material Design presentation.
The RecyclerView Library manages lists of Episodes.
The Leanback Support Library is required by Android TV.
The Espresso Library helps with creating unit tests.

Code Reuse

The Google sample for ***Android Universal Music Player (UAMP)*** has code for the *MusicService* that can be reused for this project. I plan to copy and reuse the.code related to MusicService.java and Playback.java from that Google sample.

Managed Data

Firebase will be used as the data source when initializing local SQLite tables. A Content Provider Generator (from https://github.com/BoD/android-contentprovider-generator) will create a Content Provider for access to these four logical entity groupings:

- **Configuration**
  - Firebase ID
  - Authentication
  - Android Advertising ID
  - ACCESS permissions for each show (1399 total)
  - HEARD state with timestamp for each show (1399 total)
  - PLAY_COUNT for each show (1399 total)
  - CACHED state with timestamp for each show (1399 total)

- **Episodes** (1399 total)
  - Episode Number
  - Air Date
  - Title
  - Description
  - Download URL
  - Rating
  - List of Actors in Episode
  - List of Writers for Episode

- **Actors**
  - Full Name
  - Actor's full Episode List
  - Picture (if available)

- **Writers**
  - Full Name
  - Writer's full Episode List
  - Picture (if available)

Additionally, an app managed *CACHE* directory may contain episode downloads locally. This cache directory (and any contents) must be destroyed when the app is uninstalled. The user can delete cached mp3 shows on demand from Settings. Cached shows can be played without authentication.

Next Steps: Required Tasks

Task 1: Project Setup

From Android Studio: do New Project "*Radio Mystery Theater*"

- Create using path *com.harlie.radiotheater*
- Include sub-projects for Wear, TV and Android Auto (for project structure and future use)
- Supported Minimum SDK: API 21 - Android Media Library minimum is Lollipop
- Start with the Master / Detail Flow and use "Episode/Episodes" for the Object Kind
- Add an "Always On" Wear *RadioWearActivity* (likely not implemented for Capstone 2)
- Add an Android TV *RadioTVActivity* (also likely not implemented in time for Capstone 2)
- Add Android Media Service *RadioTheaterService*
- *Create Project*
- Add Project to Github

Task 2: Create Source Datasets

Write a Python program using the *BeautifulSoup* library that functions to screen-scrape data. I need to capture Episode, Actor and Writer info from http://www.cbsrmt.com, including download links for Episode mp3 files. Save this data into JSON files that can be uploaded into Firebase as the master *Episodes*, *Actors* and *Writers* databases. Copy portrait images of Actors and Writers for inclusion in the app resources.

Task 3: Setup Firebase

Login to https://firebase.google.com: and :

- Create a Firebase account
- Create a new Firebase project: "Radio Mystery Theater"
- Use the Firebase Android "Quick Start" links to set up: *Authentication, Real Time Database, Analytics, Remote Config, Crash Reporting, App Indexing, and AdMob*. Modify the mobile *build.gradle* to include these new Firebase library dependencies.
- Firebase Auth will use: email, Google, Facebook, Twitter
- Setup a Facebook Developer Account to include Facebook AppId and support.
- Setup a Twitter Developer Account to include Twitter AppId and support.
- Upload the datasets created from Task 2 into Firebase.
- Ensure Firebase *access protections* are in place for the uploaded data.

Task 4: Create an AuthenticationActivity and an AutoplayActivity

The *AuthenticationActivity* is the app LAUNCHER Activity from AndroidManifest.xml

- First modify the app to use CoordinatorLayout + AppBarLayout + CollapsingToolbar
- Create layout XML for the AuthenticationActivity, implement portrait only
- Create layout XML for the AutoplayActivity, implement portrait and landscape
- Implement the AuthenticationActivity to invoke Firebase Authentication Login
- After successful authentication, or if already authenticated, start the *AutoplayActivity*
- Write an Android Unit Test to validate Firebase authentication.
- Write an Android Unit Test to confirm Firebase remote database access.

Task 5: Create a local database Content Provider

Generate a Content Provider. See https://github.com/BoD/android-contentprovider-generator

- Modify the mobile *build.gradle* to include new SQLite library dependencies.
- Create JSON descriptions for each dataset: Episodes, Actors, Writers, Configuration
- Create a shell script to generate a Content Provider from the JSON and run it
- Write an Android Unit Test to validate local SQLite access via the new Content Provider

Task 6: Initialize the local SQLite database

Modify the AuthenticationActivity to request and download 4 JSON databases on initial launch:

- Request Firebase JSON for Episodes, Actors, Writers and the user Configuration
- Parse JSON and insert SQLite records for: Episodes, Actors, Writers, Configuration
- Don't request JSON or user Authentication if the SQLite tables are already present.
- Write an Android Unit Test to confirm the SQLite tables are properly populated.

Task 7: Create the RadioTheaterService

Copy and refactor code from the Google Sample: *MediaBrowserService*

- Implement code related to MusicService.java as *RadioTheaterService.java*
- Implement code related to Playback.java as *RadioPlayback.java*
- Modify the RadioTheaterService to track usage with Google/Firebase Analytics.
- Modify the RadioTheaterService to track usage via SQLite *Configuration* table updates.
- Modify the AutoplayActivity to start, stop and invoke the RadioTheaterService.
- Modify the AutoplayActivity to locate the next available unheard Episode.
- Modify the AutoplayActivity to play the next Episode using the RadioTheaterService.
- Write a Unit Test to verify that AutoplayActivity playback via the big red button works.

Task 8: Create the SettingsActivity (Requirements Change due to Task 9)

Create the Settings menu:

- Create a Global Shared Preference for "*Use Large Text Font.*"
  This is for Easy Readability. The default value is ON.
- Create a Global Shared Preference for "*Clickable Episode Number.*"
  - Modify the AutoplayActivity to use the "*Episode #*" as an EpisodeDetail view access button when ON. It is initially disabled to prevent unintentional button presses.

  This enables EpisodeDetail navigation via a new button listener in the AutoplayActivity.
  The default value is off, because disabled users likely need the simplest possible UI.
- Create a Global Shared Preference for "*Download instead of Streaming.*"
  This enables the CACHE for offline listening. Episodes are downloaded first then played.
  The default value is off. Don't download to CACHE unless a user first enables this.
- Create a checkbox "button" to "*Delete Offline Downloads.*"
  This is really a button disguised as an OFF checkbox for deleting the local CACHE.
  When checked, a popup alert warns the user before deleting any downloaded episodes.
  If YES, then do the mp3 file deletions. Afterward, the checkbox returns to the OFF state.
- Create a TextView to inform users about the Search "**help**" command ability.
  A lot of power is hidden inside Search in a way that won't intimidate casual users.

Task 9: Implement Episode 'Download' functionality (no time to implement)

Until now, only "Streaming" audio has been used. Here we implement the ability to download and CACHE Episodes locally, and to play them while offline.

- Modify the AutoplayActivity and the RadioTheaterService to use the download CACHE if the Global Shared Preference value for "Use Download instead of Streaming" is set ON.
- Implement code to do the download using the *Resumable Media Download* API
- During a download operation, the AutoplayActivity displays a Circle Progress View.
- Playback begins automatically once the download completes.
- Write a Unit Test to validate proper download to CACHE.
- Write a Unit Test to validate automatic playback after download completion.
- Write a Unit Test to validate the Settings menu can delete the download CACHE.

Task 10: Create the SyncService (dependent on Task 9 - not implemented)

The *SyncService* is responsible for syncing of local SQLite *Configuration* data with Firebase.

- Run in the background and wake up when network state changes to “network available.”
- Select local Configuration records with the “*dirty*” flag set and send them to Firebase
- After Firebase is successfully updated, clear the “*dirty*” flags from Configuration.
- Write a Unit Test to validate successful sync.
- Ensure the SyncService is destroyed in Autoplay Activity’s onDestroy.

Task 11: Create a RecyclerView for viewing the Episodes list

Create the *EpisodeListActivity* and bind SQLite Episodes to it:

- Create the EpisodeList layout XML for **phones**. Implement portrait and landscape.
- Create the EpisodeList layout XML for **tablets**. Implement portrait and landscape.
- Implement the associated EpisodeListActivity.java and EpisodeViewHolder.java. The view holder must have logic to color-code CACHED and HEARD list items properly.
- Create a *RadioCursorLoader* Loader to map SQLite *Episodes to EpisodeListActivity*.
- Handle rotation events properly by repositioning the list to the previously displayed item.
- Be sure to account for the Global Shared Pref: *“Use Large Text Font*.”
- Write a Unit Test to validate proper list population and behavior on Tablet and Phone.

Task 12: Create a DetailView for viewing Episode detail

Create the *EpisodeDetailActivity* and *EpisodeDetailFragment*.

- Create the EpisodeDetail layout XML for **phones**. Implement portrait and landscape.
- Create the EpisodeDetail layout XML for **tablets**. Implement portrait and landscape.
- Episode info is passed to EpisodeDetailActivity via Parcelable. Extract and display it.
- Handle rotation events properly by using onSaveInstanceState.
- Be sure to account for the Global Shared Pref: “*Use Large Text Font*.”
- During a download operation, the EpisodeDetailActivity displays a Circle Progress View.
- Write a Unit Test to validate proper detail view population / behavior on Tablet and Phone.

Task 13: Implement SearchActivity (no time to implement)

The “Search Episodes..” functionality can understand and process voice and/or text input.

- A restricted problem domain means semantics can be parsed on the phone.
- Obtain a **key** for the Google Translate API. Implement that API in SearchActivity.
- Modify the AutoplayActivity to initialize and invoke SearchActivity for searches.
- Implement a Parser using global state to decode possible voice commands.
- Handle the “*Help*” command: just open http://harlie.com/radiotheater/help.html
- Generate an appropriate SQL query from translated or typed text and execute it.
- Display the returned selection results in the RecyclerView as an EpisodeList.
- Write a Unit Test to validate canned searches.

Task 14: Final Cleanup and Implement Free and Paid versions

Implement code trimmings and **purchasing**. Custom Gradle rules allow #IFDEF ‘buildType’ to create both free and paid versions from the same source code.

- Construct a home screen Widget with the big red Play/Pause/Resume button on it.
- Implement the Sharing Button to email friends about an episode.
- Integrate the ‘preprocessor.gradle’ with the build.gradle.
- Finish app styling and use Material Design transitions.
- Finish any remaining code implementation related to: *Analytics Reporting, Firebase, Crash Reporting, App Indexing, and AdMob*.
- Use the ‘preprocessor.gradle’ **#IFDEF ‘TRIAL’** directive for single codebase integration.
- Implement code signing using a private keystore used by the mobile/build.gradle.
- Implement *proguard-rules.pro* to optimize apk generation.
- Build and test both TRIAL and PAID signed apk versions of *Radio Mystery Theater.*
- Submit this project including signed apks to Udacity staff for review.
- Make use of the Firebase testing utilities and services for further testing and validation.
- Upload the signed Radio Mystery Theater apps to the Google Play Store.
- Validate both apps again once they are accepted.

Future

Implement *download functionality* for offline listening capability.
Will design and implement depending on available time. Plans for expansion include:

- Android Wear integration
- Android TV integration
- Android Auto integration
- Shared experience by listening together, and group chatting
- Micro payments, per show or set of shows
- Expanded show ratings and reviews

Submission Instructions

1. After you've completed all the sections, download this document as a PDF [File → Download as PDF]
2. Create a new GitHub repo for the capstone. Name it "**Capstone Project**"
3. Add this document to your repo. Make sure it's named "**Capstone_Stage1.pdf**"