How to Use this Template

1. Create a new document, and copy and paste the text from this template into your new document [Select All → Copy → Paste into new document]
2. Name your document file: "**Capstone_Stage1**"
3. Replace the text in green

Description
Intended User
Features
User Interface Mocks
- Screen 1
- Screen 2
- Screen 3
- Screen 4
- Screen 5
- Screen 6
- Screen 7
- Screen 8
- Screen 9
- Screen 10
- Screen 11

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: Fetch data from iTunes Search API
- Task 4: Set Up the Search Interface
- Task 5: Implement the Android Architecture Component
- Task 6: Implement media streaming with ExoPlayer
- Task 7: Implement a splash screen with an AsyncTask
- Task 8: Implement a Homescreen Widget
- Task 9: Implement Google Play Services
- Task 10: Handle Error Cases
- Task 11: Include support for Accessibility and Localization
- Task 12: Add Visual Polish and Use Material Design
- Task 13: Add a Signing Configuration
- Task 14: Build an App

Task 15: Conform to common standards

GitHub Username: kimycs

Candy Pod

Description

Candy Pod is a podcast player for podcast lovers who want to listen to podcasts for free. Discover popular podcasts and listen to your favorite podcasts anywhere, anytime. It is easy to use and you can subscribe to the podcast in one click.

Intended User

This app is for anyone who loves to listen to the Podcasts.

Features

- Search podcasts by keywords or name
- Play favorite episodes
- Stream or Download episodes
- Share podcast URL with friends
- Control playing from a Homescreen Widget

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.

Screens are created by using a ninjamock.

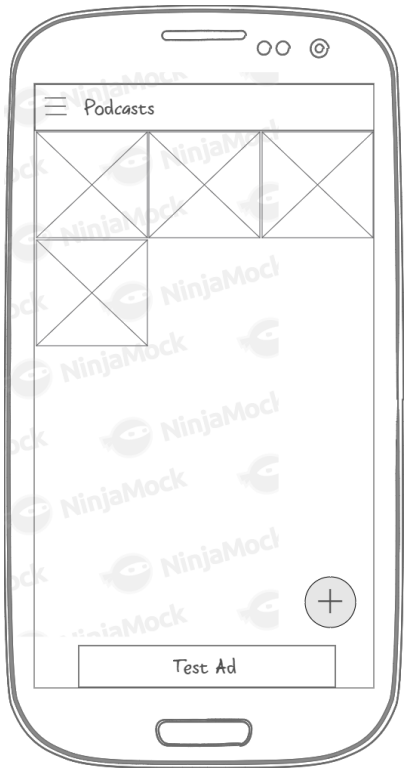
Here is a link to preview a prototype which is created by using fluidUI.

https://www.fluidui.com/editor/live/preview/cF9vSW1yZ3I1UzJvSFJ5OUxpQmJDVkhCMEFGeERSUk0zUQ==

Screen 1

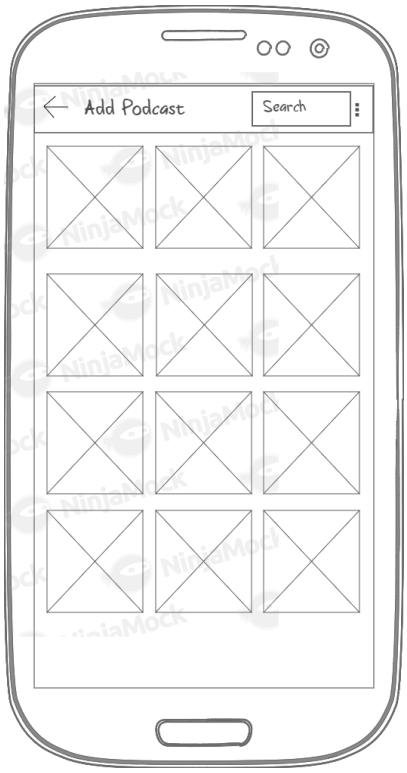
The **Navigation Drawer** shows the app's main navigation menu. The menu items will be composed of Podcasts, Playlists, Favorites, and Downloads.

Screen 2



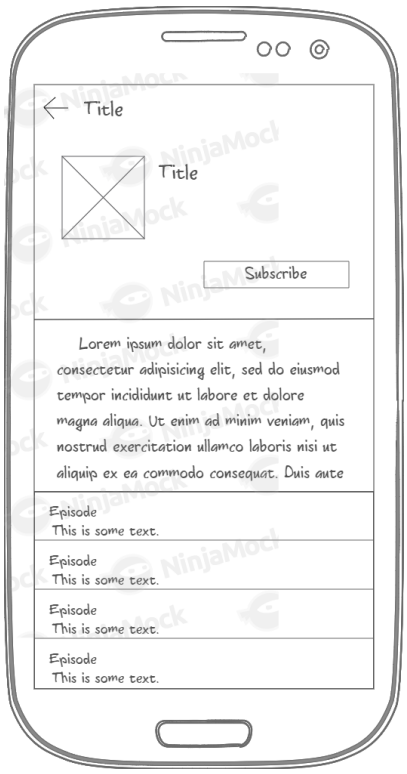
The **First Fragment** will display the list of podcasts that appear as a grid of images. At first, there will be no list of podcasts. When a user clicks a Floating Action Button on the bottom-right corner of the screen, it will launch Add Podcast Activity (Screen 3) where the user can find and select the podcast. If the user chooses the podcast and subscribes it, this fragment will display the list of podcasts. And there will be test Ad on the bottom of the screen in the free version.

Screen 3



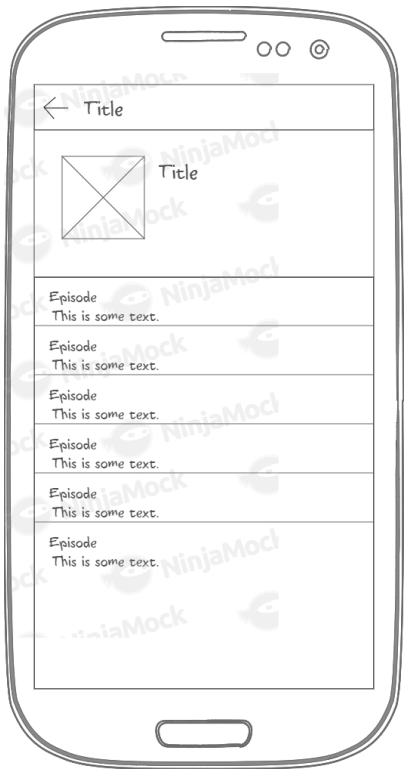
The **Add Podcast Activity** will display the list of Podcasts. The iTunes Search API will be used to fetch the podcasts data. There will be a search menu button in the app bar, which allows the user to search podcasts based on the query. When the user clicks one of the podcasts in this screen, it will navigate to the Subscribe Activity (Screen 4).

Screen 4



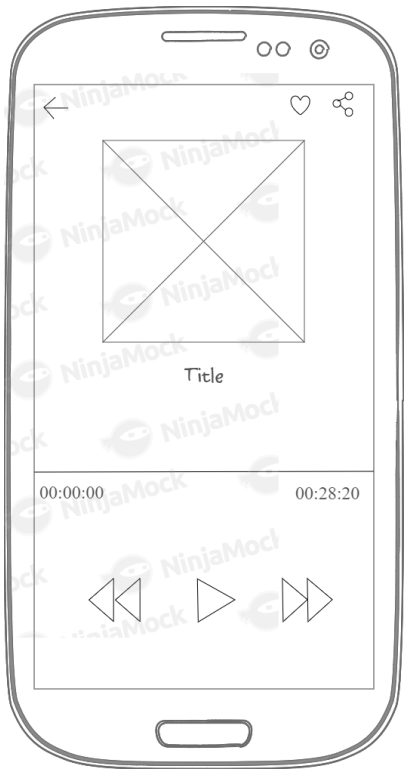
The **Subscribe Activity** will display the detailed information of the podcast such as the image, subscribe button, descriptions, and the list of episodes. If the user clicks the subscribe button, the First Fragment will show this podcast with a grid arrangement of the image (Screen 2).

Screen 5



When the user clicks on the podcast in the First Fragment (Screen 2), this **Detail Activity** will display episodes of podcasts. When one of the episodes in the list is selected, the Now Playing Activity (Screen 6) will be launched.

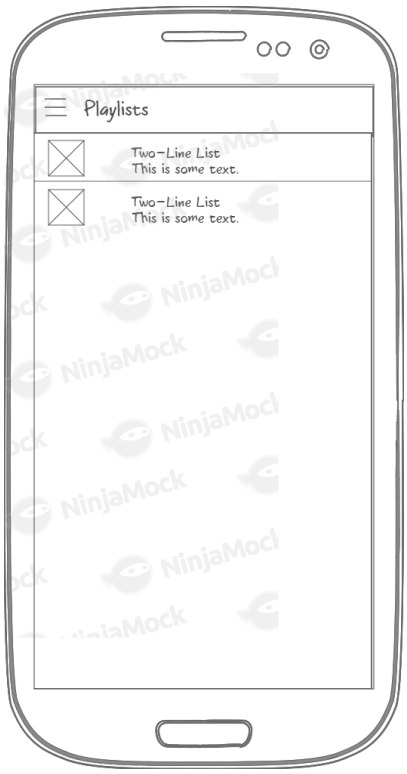
Screen 6



The **Now Playing Activity** will display the current episode, which will contain play/pause, rewind, fast forward buttons, and time bar. There will be a share button which allows users to share the episode URL. Users will be able to mark an episode as a favorite by tapping a favorite button.

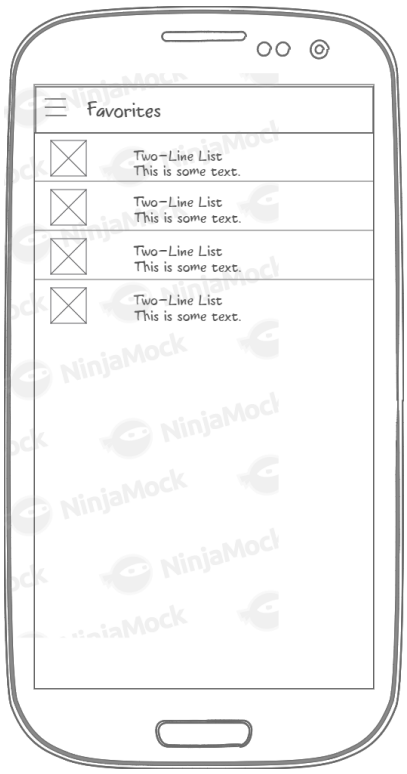
Capstone_Stage1

Screen 7


Playlists
Two-Line List
This is some text.
Two-Line List
This is some text.

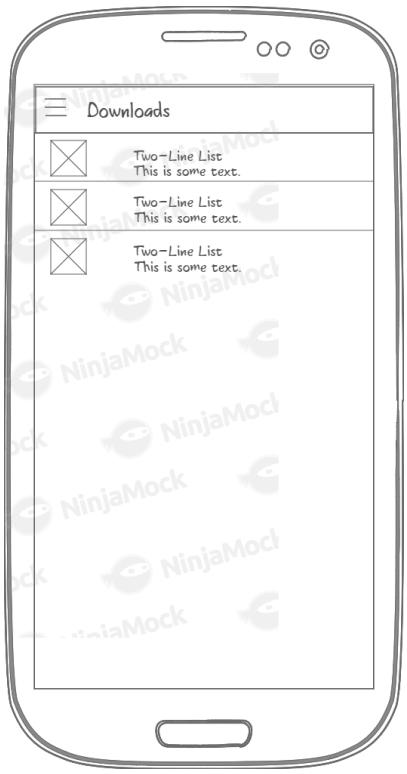
The **Second Fragment** will display the playlists which will consist of episodes that the user played.

Screen 8


Favorites
Two-Line List
This is some text.
Two-Line List
This is some text.
Two-Line List
This is some text.
Two-Line List
This is some text.

The **Third Fragment** will display the favorites collection stored in the database. The list item will contain the image, title, and length of an episode.

Screen 9


Downloads
Two-Line List
This is some text.
Two-Line List
This is some text.
Two-Line List
This is some text.

The **Fourth Fragment** will display the downloaded episodes. The list item will contain the image, title, and length of an episode.

Screen 10

The **widget** will keep the user informed about what episode is currently playing. It will contain the image, episode title, play/pause button, and fast forward button.

Screen 11

The **notification** with play/pause and fast forward action will depend on the current MediaSession PlaybackState.

Key Considerations

How will your app handle data persistence?

The app will store data locally by using Room. There will be three tables - playlist, favorites, downloads. The title, id, the length of the episode, the image URL and published date will be the column in the database.

Describe any edge or corner cases in the UX.

The user will navigate between fragments such as Podcasts, Playlists, Favorites, Downloads by using a navigation drawer.

The user will return to the Add Podcast Activity by hitting the Floating Action Button in the first fragment. The user will navigate to the Subscribe Activity by clicking the podcast item in the Add Podcast Activity.

The user will return to the Detail Activity by clicking the subscribed podcast item in the first fragment. The user will return to the Now Playing Activity by clicking the episode in the Detail Activity.

The user will return to the previous screen by hitting the back button. The up button in the app bar will allow the user to navigate to the previous screen.

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

The app will utilize stable release versions of all libraries.

- Android Support Library v7 appcompat library to support for the Action Bar, AppCompatActivity. v7 recyclerview library provides support for the RecyclerView widget. v7 cardview library to support for the CardView widget.
  ConstraintLayout to build a responsible UI.
  Design Support library provides CoordinatorLayout, AppBarLayout, CollapsingToolbarLayout, FloatingActionButton, and Snackbar.

```
implementation 'com.android.support:appcompat-v7:28.0.0'
implementation 'com.android.support:recyclerview-v7:28.0.0'
implementation 'com.android.support:cardview-v7:28.0.0' implementation
'com.android.support.constraint:constraint-layout:1.1.3' implementaition
'com.android.support:design:28.0.0'
```

- Android Architecture Components (Room, ViewModel, LiveData)
  Room Persistence Library provides an abstraction layer over SQLite to allow for more robust database access while harnessing the full power of SQLite. Room will map our database object to Java object.
  ViewModel to cache data that needs to survive configuration changes.
  LiveData makes it easy to keep what's going on screen in sync with the data.

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

implementation 'android.arch.lifecycle:extensitions:1.1.1'
annotationProcessor 'android.arch.lifecycle:compiler:1.1.1' ●
        Android Data Binding to bind UI components in the layouts to data
sources in the app.
```

```
android {
    dataBinding.enabled = true
}
```

- Glide to handle the loading and caching of images.

```
implementation 'com.github.bumptech.glide:glide:4.8.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.8.0'
```

- Retrofit for REST API communication.

```
implementation 'com.squareup.retrofit2:retrofit:2.4.0' implementation
'com.squareup.retrofit2:converter-gson:2.4.0'
```

- Gson to convert Java Objects into their JSON representation.

```
implementation 'com.google.code.gson:gson:2.8.2'
```

- ExoPlayer to play audio and video locally and over the Internet.

```
implementation 'com.google.android.exoplayer:exoplayer:2.8.2'
```

- Timber is a logger with a small, extensible API which provides utility on top of Android's normal Log class.

```
implementation 'com.jakewharton.timber:timber:4.7.1'
```

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

The App will use **Google Mobile Ads** and **Firebase Analytics**. For Google Mobile Ads, banner ads will be used on the bottom of the screen. For Firebase Analytics, initialize analytics and use analytics to log custom events.

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 project and add a navigation drawer activity. Add dependencies to the dependencies block in the build.gradle file for the module's directory.

- App is written solely in the Java Programming Language.
- App utilizes stable release versions of all libraries, Gradle, and Android Studio.
- Configure libraries
- Create a navigation drawer

Task 2: Implement UI for Each Activity and Fragment

Create each Activity and Fragment. Create layouts for multiple Activities and Fragments.

- Build UI for MainActivity
- Build UI for AddPodcastActivity
- Build UI for SubscribeActivity
- Build UI for DetailActivity
- Build UI for NowPlayingActivity
- Build UI for each Fragment

Task 3: Fetch data from iTunes Search API

Add the Internet permission in AndroidMenifest.xml to connect to the Internet. To fetch podcasts, use the iTunes Search API.

In order to request top podcasts in the US and return the first 25 items, the URL will look like the following:
https://rss.itunes.apple.com/api/v1/us/podcasts/top-podcasts/all/25/explicit.json

Extract the podcast **id** from this request and use this in subsequent requests.

To extract mp3 URL for the podcast, use the podcast id and create a lookup request to search for the podcast. For example, to search for the Bag Man podcast, the URL will look like the following:
https://itunes.apple.com/lookup?id=1438463967
Then, find the "**feedUrl**" from the request and parse the RSS feed it contains. That will have the episode metadata and stream URLs for the audio file.

- Add the Internet permission
- Work with the iTunes Search API
- Use Retrofit to turn HTTP API into a Java interface
- Create a POJO
- Parse an RSS feed

Task 4: Set Up the Search Interface

- Add the Search View to the app bar in the Add Podcast Activity

Task 5: Implement the Android Architecture Component

Create three database tables for saving playlists, favorites, downloads. The title, podcast id, the image URL, the length of an episode will be the column of the table.

- Create an Entity to define a database table
- Create a DAO to provide an API for reading and writing data
- Create a Database which represents a database holder
- Add the LiveData and ViewModel
- Make sure no unnecessary calls to the database are made

Task 6: Implement media streaming with ExoPlayer

Use ExoPlayer to play an episode. Initialize and release audio assets when appropriate.
Show Media Style notification, with actions that depend on the current Media Session PlaybackState.

- Initialize ExoPlayer
- Create a MediaSource
- Release ExoPlayer
- Restore the playback position
- Customize the PlayerControlView
- Set the Player.EventListener
- Add Media Session and Media Notification

Task 7: Implement a splash screen with an AsyncTask

- Create a new Activity for a splash screen
- Load a splash screen before entering the main screen by using an AsyncTask

Task 8: Implement a Homescreen Widget

App provides a widget to provide relevant information to the user on the home screen.

- Create WidgetProvider
- Create the App Widget Layout

Task 9: Implement Google Play Services

- Display test ads for a free version
- Connect the app to the Firebase
- Each service imported in the build.gradle is used in the app. ● App creates only one analytics instance.
- Add Log events for the Firebase Analytics

Task 10: Handle Error Cases

- Handle errors when loading images by using .error() method of Glide
- Display an empty screen when the user does not have any subscriptions
- Check network connectivity. If there is no network connectivity, show a snackbar message
- App validates all input from servers and users. If data does not exist or is in the wrong format, the app logs this fact and does not crash.

Task 11: Include support for Accessibility and Localization

Describe all ImageViews, ImageButtons and all Checkboxes using the content description attribute. Move all the hardcoded Strings to the strings.xml file and implement support RTL layouts.

- Add content descriptions, navigation using a D-pad
- Keep all strings in a strings.xml file
- Enable RTL layout switching on all layouts

Task 12: Add Visual Polish and Use Material Design

- Change colors and fonts
- Add a touch selector
- Make the app more delightful with material design patterns
- App uses standard and simple transitions between activities
- App theme extends AppCompat
- App uses an app bar and associated toolbars

Task 13: Add a Signing Configuration

- Create a keystore and a key
- Create a signing config in build.gradle
- Assign the signing configuration to a build type
- The keystore and passwords are included in the repository. Keystore is referred to by a relative path.

Task 14: Build an App

- App builds from a clean repository checkout with no additional configuration.
- App builds and deploys using the installRelease Gradle task.
- All app dependencies are managed by Gradle.

Task 15: Conform to common standards

- App conforms to common standards found in the Android Nanodegree General Project Guidelines

---

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**"