## Android Fundamentals Project Self-Evaluation

**Instructions:** Once you’ve completed your Final Project, please evaluate it against the components of the rubric below. For each criteria that you met, put an “X” in either the “Does Not Meet Specifications” or the “Meets Specifications” box. For some criteria, we ask you to provide an explanation of where and how it was implemented in your app. This is a chance for you to briefly explain to the grader your thought-process during development. Once you are done, include this with the source code and accompanying files you are submitting. Then, give yourself a pat on the back for making a great app!

### Required Components

To “meet specifications”, your app must fulfill all of the criteria listed in this section of the rubric.

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Does Not Meet Specifications** | **Meets Specifications** |
| Standard Design |  |  |
| App does not redefine the expected function of a system icon (such as the Back button). |  | x |
| App does not replace a system icon with a completely different icon if it triggers the standard UI behavior. |  | x |
| App does not redefine or misuse Android UI patterns, such that icons or behaviors could be misleading or confusing to users. |  | x |
| App includes a tablet layout which takes advantage of the additional space (if possible). |  | x |
| App includes at least two distinct views and uses intents properly to move between these views. |  | x |
| **Navigation** |  |  |
| App supports standard system Back button navigation and does not make use of any custom, on-screen "Back button" prompts. |  | x |
| All dialogs are dismissible using the Back button. |  | x |
| Pressing the Home button at any point navigates to the Home screen of the device. |  | x |
| **Permissions** |  |  |
| App requests only the absolute minimum permissions that it needs to support core functionality. |  | x |
| App does not request permissions to access sensitive data or services that can cost the user money, unless related to a core capability of the app. |  | x |
| **Please elaborate on why you chose these permissions:**  Only *android.permission.INTERNET* can cost money to the user, as the application may connect over a metered network. |  |  |
| **Performance and Stability** |  |  |
| App does not crash, force close, freeze, or otherwise function abnormally on any targeted device. |  | x |
| **ContentProvider** |  |  |
| App implements a ContentProvider to access locally stored data. |  | x |
| 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.  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. |  | x |
| App uses a Loader to move its data to its views. |  | x |
| **1) What's the content provider called, and how is it backed?**   * *PodcastContentProvider*, backed by an SQLite database. There are three main tables, Podcasts, Episodes and Playlist. Playlist data is organized as a doubly-linked rows to allow easier reordering of elements.   **2) What backend does it talk to? What is the SyncAdapter called? What mechanism is used to actually talk over the network?**   * Initial podcast data (like subscriber count, RSS feed urls, etc) is fetched from GPodder’s public API. * *PodcastSyncAdapter* * RSS feed data, image downloading, media streaming and API calls are done through HTTP. The application uses a single instance of Square’s OkHttpClient to manage any network requests. There is also a disk response cache to aid loading times and to minimize the network usage as much as possible.   **3) What loaders/adaptors are used?**   * App’s architecture makes Loaders quite unnecessary. Most data deliveries are centered around the idea of a push-based architecture, where observers get notified about data changes and refresh the dataset. Any non-trivial screens have a separate presenter class dedicated to loading and delivering data to the associated activites/fragments/views. Most asynchonous tasks are implemented in the form of RxJava Observable streams. Not only that they are much more flexible, but also have one notable feature that the Loader framework is lacking, which is simple, understandable and easy \*\*\*Error Handling\*\*\*! * Most database queries are done with [SQLBrite](https://github.com/square/sqlbrite), a SQLiteDatabase/ContentResolver wrapper that enables the data loading and changes observation to be done by the means or RxJava Observables. Any emissions can be mutated, filtered, cached, delayed, … which gives total freedom to how and when the data can be consumed. * The application is built around [Nucleus](https://github.com/konmik/nucleus), a tiny library adding MVP-pattern enabled architecture to Android projects. It adds a Presenter middleman between Views(Activities, Fragments, View/ViewGroups) and the model layer. The library adds a few very clever RxJava operators designed to aid the data deliveries to the views while letting through transmissions only while there is a view attached. It also has utilities for automatic restarting of asynchronous, long running tasks on screen state restoration. More details about the library’s feature can be found in the link above. * Given the fact that having a Loader is necessary, there is one named *AppVersionLoader* in Settings-> About which loads the app version name and version code asynchronously just to meet this controversial requirement. |  |  |
| **User/App State** |  |  |
| App correctly preserves and restores user or app state. |  | X |
| When the app is resumed after the device wakes from sleep (locked) state, the app returns the user to the exact state in which it was last used. |  | X |
| When the app is relaunched from Home or All Apps, the app restores the app state as closely as possible to the previous state. |  | X |
| **Please elaborate on how/where your app correctly preserves and restores user or app state:**  Most of the data that has to be displayed is taken from the ContentProvider and is subject to changes at any time.  The View Presenters aim to preserve the state (mostly Model-related objects in adapters) during orientation changes, while the SQLite-backed ContentProvider takes care of the cases where the app’s process gets restarted/cleaned. See the ansers about the Loaders in the previous section, more information on how the exact View < ---> Presenter interaction takes place can be found in [Nucleus's Github page](https://github.com/konmik/nucleus). |  |  |

### 

### Optional Components

To receive “exceeds specifications”, your app must fully implement all of the criteria listed under at least two of the four categories below (e.g. Notifications, ShareActionProvider, Broadcast Events, and Custom Views).

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Does Not Exceed Specifications** | **Exceeds Specifications** |
| Notifications |  |  |
| Notifications do not contain advertising or content unrelated to the core function of the app. |  | x |
| Notifications are persistent only if related to ongoing events (such as music playback or a phone call). |  | x |
| Multiple notifications are stacked into a single notification object, where possible. |  | x |
| App uses notifications only to indicate a context change relating to the user personally (such as an incoming message). |  | x |
| App uses notifications only to expose information/controls relating to an ongoing event (such as music playback or a phone call). |  | x |
| **Please elaborate on how/where you implemented Notifications in your app:**  There is a Media-Style notification for the playback controls. |  | x |
| **ShareActionProvider** |  |  |
| Uses ShareActionProvider to share content with an outside application. |  | x |
| Makes use of Intent Extras to send rich content (i.e. a paragraph of content-specific text, a link and description, an image, etc). |  | x |
| **Please elaborate on how/where you implemented ShareActionProvider:**  There is an share menu item in the Player’s screen. It is located in the overflow menu of the sliding panel.  The menu item is a ShareActionProvider, which sends the current episode’s content URL. |  | x |
| **Broadcast Events** |  |  |
| App intercepts broadcast events. |  | x |
| App responds to Broadcast events in a meaningful way. |  | x |
| **Please elaborate on how/where you implemented Broadcast Events:**  App Widget’s click event handling, DownloadManager’s state change intents, MediaSessionCompat’s media button events and maybe something else |  | x |
| **Custom Views** |  |  |
| App creates and uses a custom View. |  | x |
| App uses a novel View that couldn’t sufficiently be satisfied by the core Views in Android. |  | x |
| **Please elaborate on how/where you implemented Custom Views:**  AspectRatioImageView is an ImageVIew-derived view which can show Images while maintaining a fixed aspect ratio. Used all over the app. |  | x |