Assessment project: <https://bit.ly/2RjF8nT>

Outcome Health Android Team Code Challenge

To successfully complete this challenge, you must build a working Android video gallery and player application that fulfills the following requirements within the timeline specified when you received the assignment, then submit a zip archive of your completed project.

* Code should be well-structured and reasonably documented
* Open-source libraries are permitted
* Project must support devices back to Android Kitkat
* Either fetch and parse video manifest file, or package the file with your app
* Display the contents of the manifest in a clickable gallery, where each entry is comprised of a video thumbnail and title
* In portrait mode, display a selected video above the gallery, i.e. YouTube in portrait mode
* In landscape mode, videos must play fullscreen
* Video playback position must be maintained through orientation changes
* The next video in the list should auto-play after first user selection completes
* Bonus: Videos can play offline
* Bonus: Each gallery entry displays video duration alongside title and thumbnail
* Bonus: Write an instrumentation test for part of the app

**Note: The requirements are intentionally left open-ended, allowing freedom to choose specific design and implementation details, as long as they still ultimately satisfy the requirements above. Be prepared to explain your choices.**

Brief Coding Walkthrough

1. **TheApp** is an Application-extended object which is a single and is instantiated prior to any other class when the application/package is created. This class is primarily used for initialization of global state before the first **Activity** is created.  
   There are a few acceptable uses of a custom application class:
   * Specialized tasks that need to run before the creation of your first activity
   * Global initialization that needs to be shared across all components (crash reporting, persistence database, etc.)
   * Static methods for easy access to static immutable data such as a shared network client object
2. **MainActivity** is the startup activity to launch, **onCreate()** callback, it will attach the **MainFragment** fragment to this activity.
3. **MainFragment.onCreate()** Will be invoked afterwards, and it will read and parse the video data stored in the JSON file “**videos.json**” which is in the app’s resource asset **raw** folder. The parsed video data will be stored in the **MutableLiveData<VideoData>-derived** object.
4. **MainFragment.onCreateView()** will set the size of the **VideoView** video player and the **RecyclerView** list view appropriately at runtime to accommodate different Android device screen size and resolution.
5. A **RecyclerView.Adapter<VideoItemRecyclerViewAdapter.ViewHolder>-derived** class

**VideoItemRecyclerViewAdapter** binds the List<VideoData> list to the **RecyclerView**. **VideoItemRecyclerViewAdapter.onBindViewHolder()** will bind each VideoData to the ViewHolder. During data binding, if the time duration is not set, it will retrieve the time duration from the video (works for both online and local video).  
  
*Note that retrieving the time duration will cause the app to run a bit slower. However, once time duration of all the videos have retrieved and cached the app will run smoothly and swiftly. There is a checkbox in the Settings page to allow the users to enable retrieving time duration or not.*

1. **MainViewModel** is a **ViewModel-derived** object, which contains a **VideoData** object that the app is used to display the video. In **MainFragment.onActivityCreated()**, it observes this **VideoData** object, and whenever this **VideoData** is changed or modified, then the **Observer<VideoData>.onChange(VideoData videoData)**will be invoked.
2. Therefore, when MainViewModel.setVideoData(VideoData videoData) sets a different VideoData object to play, the observer **Observer<VideoData>.onChange(VideoData videoData)** (in **MainFragment.onActivityCreated()**)will be called automatically to play the newly selected video from beginning.
3. When the video ends, the **VideoView.setOnCompletionListener**’s **onCompletion(MediaPlayer mp)** will be called, and the next video will be selected. Once the end of the video list has reached, it will loop back to the first video in the list.
4. The video in the list can also be selected randomly from the gallery list view.