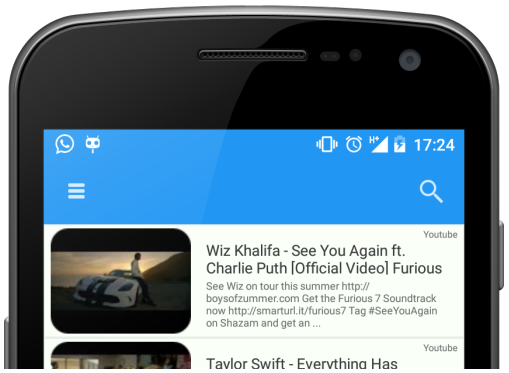
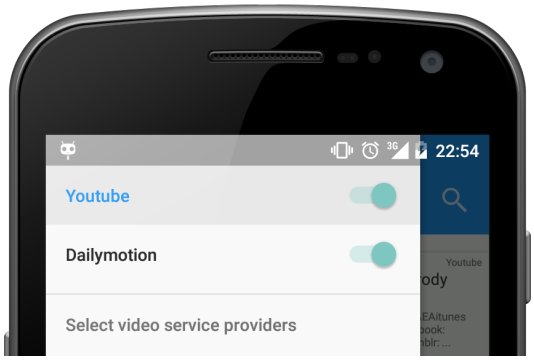
Video ping

Video ping is an app developed for the CroApps 2015 contest.

Lets talk design

The application provides a simple interface for searching and viewing content from various video providers online.
Video ping takes into consideration googles official material design guidelines for providing a simple yet intuitive interface.Taking into consideration older users of the Android platform, Video ping is made using the AppCompat v7 library so It can provide compatibility among wide variety of Android versions. Among other thing is uses the new Toolbar widget which replaces the existing ActionBar - providing the same functionality but more customisability for the programmer. Animations, Drawers and Material designs - It's all there.





Libraries used in the project

AppCompat v7 - googles official library that allows backward compatibility for the new material design
RecycleView v7 - part of AppCompat v7
MaterialDrawer 2.9.7 - excellent library for sweat-free implementation of the navigation drawer, with superb documentation and vide variety of pre-provided elements
Google Youtube API v3 - API library needed for connecting to the Google servers
Json - lightweight data exchange format
Google query 0.25.9 - great library providing all the code needed for grabbing images from links. It also provides great memory management system and view animations
SmoothProgressBat 1.1.0 - visual sugar that is cross SDK fully compatible and provides the user with feedback about internet fetching and processing
Joda-Time 2.7 - time library to take care of the youtube ISO compatible timestamp, and similar

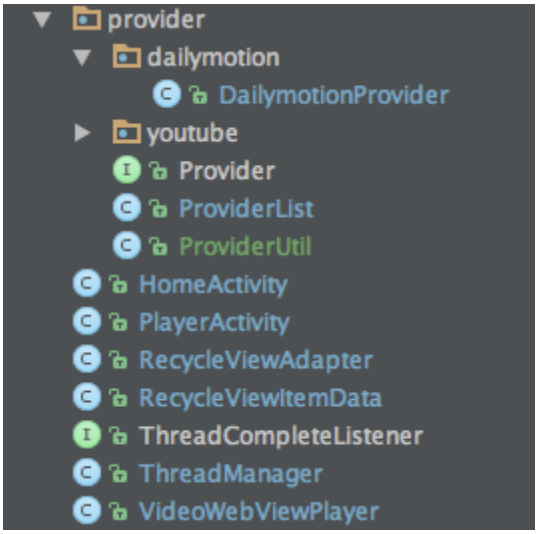


Figure 1.1

So… how do I add new providers?

Video ping comes with 2 interfaces and 7 classes(figure 1.1) + provider classes. For a programmer to add a new service to the application there are only two thing to take into consideration:

1) your provider object needs to implement all methods from the ‘Provider’ interface
2) A new instance of the provider object should be enlisted in the ’ProviderList’ object, in the ‘listOfAllProvides’ array (check ProviderList.java for more information)

The ‘Provider’ interface(figure 1.2) requires the implementation of only the most necessary methods. Methods worth noting are the ‘fetchDataFromServer’ and ‘getFullVideoUrl’.

```
package com.example.filipsaina.videoping.provider;

import ...

/**
 * Interface that every provider class must fulfill.
 * Created by filipsaina on 15/05/15.
 */
public interface Provider {

    public String getProviderName();
    public List<RecycleViewItemData> fetchDataFromServer(String searchTerm);
    public String getFullVideoUrl(String videoId, String startTime);

}
```

Figure 1.2

The first one should contain all the necessary code that handles the connecting(http) and parsing the return data in the return format.

The data returned is a List of RecycleViewItemData items that should be declared within the method and put into a ArrayList(preferably). Necessary to mention is that this method(fetchDataFromServer) is going to be executed within its own Thread so the programmer does not need to worry about performance issues or defining his own Runnable or Thread object. The later method provides the method caller with a String object containing the full link to the embedded video for the videoId(see provided code for examples).

Currently the data required to create each RecycleViewItemData Object(Figure 1.3) is quite simple, but Its easily extendable(providing some additional declarations in the .xml layout file and the RecycleViewAdapter Class)

```
public class RecycleViewItemData {

    private String videoTitle;
    private String videoId;
    private String imageURL;
    private String videoDescription;
    private int duration;
    private int providerIndex;
```

Figure 1.3

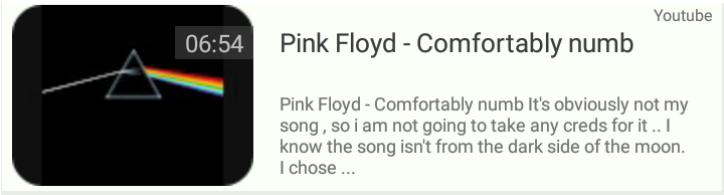


Figure 1.4 - an example of a search result in the RecycleView

So now, the programmer has implemented the necessary methods and did all the code within. In order for the class to be executed within the code, it is necessary to include a new instance of the Object in the listOfAllProviders Providers array(Figure 1.5).

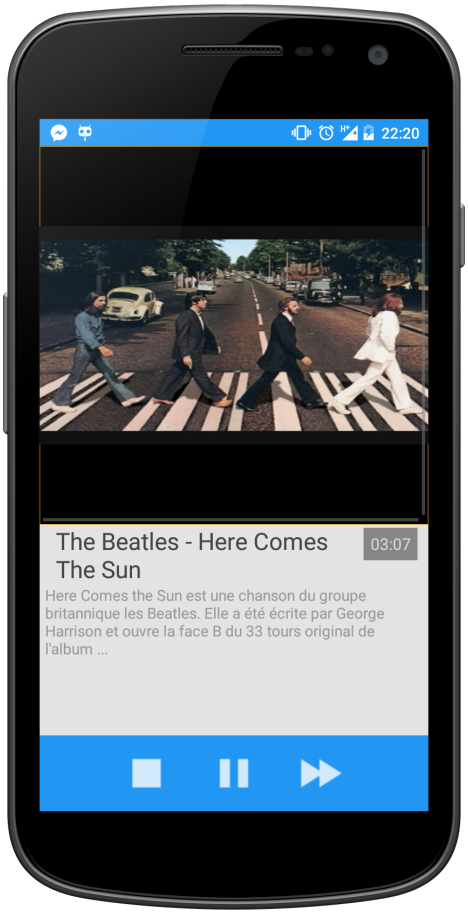
And that's it, in the application now you should see a new item in your drawer and be able to perform search queries on your favourite provider service.

```
private static Provider[] listOfAllProviders = {
        new YoutubeProvider(),
        new DailymotionProvider()
};
```

Figure 1.5

Play. Stop. Pause.

Video ping uses a extended WebView component for previewing media content. For more information check out the VideoWebViewPlayer class. It is a preset WebView component that comes with controller methods.
Additionally, the webView catches all user input and simply ignores it - now the user is obligated to use the outside controls.
Video should autostart on the activity call - this is achieved by simulating a user press every time the WebView completes loading the page(so is the play/pause calls). That way, regardless of what provider we implement, it is bound to provide the same playback features.
(note: The defined behaviour for the user interaction calls (onBackPressed, etc.) is defined in the PlayerActivity class, not in the VideoWebViewPlayer class)



Note:
Currently implemented video providers are Youtube and Dailymotion. While Youtube provides full targeted functionality, the Dailymotion lacks the 'seek to' option. The reason is that Dailymotion API does not provide a way of controlling media playback in such a manner and should be taken into consideration by the reviewers.
For more information on the subject visit:
https://developer.dailymotion.com/player#player-parameters

Important:
Running the .apk in a virtual machine may cause unexpected behaviour. Please run it on a mobile device. Minimal required SDK version 15(Android ICS 4.0.3).