#### AppKit Master Document

Section Index:

*~~Setup~~*

* Name Change Steps
* Initiate Google Services
* Analytics
* AdMob
* ChatGPT

~~*Usage~~*

* Debug Fragment
* Network
* Database
* Settings
* Intro
* Permissions
* Location
* Dialogs
* BottomSheets
* Navigation
* Asset Loader
* Utilz, DateHelper
* ErrorController
* Dark Theme

# Architecture

The app has a MVVM design to it, it also uses Google’s navigation system making it easier to build multi-tabbed apps. The core is the glue between them all, although you can access the network and database directly, you could also make the core module do all the logic (I leave the choice up to you). All the functionality is based off the same principle, the client is the service, the controller holds the public functions, and the interface is used as the callback/listener to the services.

# Initial Setup

I recommend making a copy of the folder into its final destination, if using Git then clone the repo onto your machine first and then place the contents of appkit/ into the new repo. Load it into Android Studio and build the application. Check out the DevDocs.kt class for a quick setup or keep reading for a more detailed version.

#### ~~SETUP~~

# Name Change

First thing I do is change the name, it is easier when there are less files as you will have to open each file to auto import the correct path after renaming it. (Auto import can be found by searching for import in the settings, enable both import and optimize on the fly.

1. Change tree appearance to NOT compact middle packages, this will allow us to select the package ‘appkit’.
2. Select the ‘appkit’ package and right click to rename it. (reformat -> rename)
3. Select All in Current Module.
4. Rename package name
5. Select All places in the Scope.
6. Run refactor.
7. Open build.gradle (:app) change namespace and application id to the new name.
8. Open settings.gradle and change the rootproject.name to the new name.
9. Build, we should see errors about an import from appkit... This is life.
10. Open each class in the app module (19 only), delete the incorrect imports and allow to auto import and optimize the class on the fly.

Now Invalidate Caches and restart (no need to select any of the additional options). Once completed run the app and you should see your new name show up next to the Appkit app (assuming you already ran it before this point)....it’s ready now.

# Google Services

If you want to use any of Firebase or Google services we need to setup a project on Firebase.com. Luckily everything is setup for you already, all you will need is the google-services.json file and uncomment some lines to get it running.

Place the google-services.json file in the proper location in the app module, then in the build.gradle (:app) uncomment the plugins for google-services. You can also uncomment the others at this point. We also need to uncomment the initialization line in the MainActivity (line 74 in the onCreate function). To test that everything works, go to the DebugFragment.kt class and at the top there is a var named enableGoogleSerivces, it is currently set to false, switch to true and run the app. If it does not crash, it worked.

# Analytics

Additional features available to use right out of the box are Firebase Crashlytics, Analytics, Performance, Authentication, Billing, and can easily add more following their documents to enable them in the Play Console. You may have to provide a sha-1 key to connect them to your app.

# Admob

Admob is also integrated, although an Admob account is required. Follow their documentation on how to setup an app through their panel. Add the app id to the AndroidManifest (line 40, where APPLICATION\_ID is). ie: android:value=”place here”. The app is currently setup to run interstitial ads but can easily run rewared ads as well. Currently the id’s for the unit ad is located in the DEV\_DOCS.kt class for easier explaining. Here you will find a test id as well that you SHOULD use while in debug, the app is already setup to do this.

# ChatGPT

The app comes ready to ask OpenAI’s chatGpt questions, this includes a wrapper for the params and all! To set this up go to openai.com and create an account to get a api key. Place this key in the NetworkConstants class located in the network module, ie. "Bearer key-right-here" keeping the same format. You can test this in the debug tab -> chat gpt section.

#### ~~USAGE~~

# Debug Fragment

This class has all the test functions to test out the pre-baked goodies, it is also a place where you can easily test out new features you implement. You can easily disconnect it from the app by simply removing it from the menu of the bottom bar. I encore you to use this class to test out the Google services features using this class.

# Adding a new Fragment

To add a new fragment you would first add the .kt class and the .xml layout for the class, then locate the main\_nav\_graph.xml file located in the nav resource package. In the upper left side of the design window you will see a green plus sign in front of a rectangle. To connect it to an existing fragment add the action tag in the host’s fragment tag seen on the file already. You can use the animations that are in the anim to get the right fit for what you are building.

# Network

The network module holds all the logic for making your api calls easily, the client is where the heart of the service is. You have the option of using a client with headers or without, these can be found in the NetworkClient class of the network module. You can even use the Bearer method as used with chatGpt api calls. To add more calls add a function in here and connect it with the NetworkController class. Here in the controller you can build your ‘payload’ or params as some call it. This class holds the retry logic as well as the interface for the response call, don’t forget to destroy the connection in the event the user presses the home button. A simple example is found in the DebugFragment, it calls the countries api to pull a list of countries and display the amount on the screen.

# Database

The app is already wired to use Google’s Room Database, you can easily build upon the existing code to add new tables. Similarly designed as the network module, the AppDatabase is the ‘client’ where it holds all the logic for the service. This class unlike the network needs a Dao to operate. The Dao will hold the SQL logic if you desire any, some examples are there using the countries api data. The controller is where it the data is added to the database, this is done on a different thread. This uses an interface to inform the main thread it has completed the write. To read from the database I use a Disposable on a Schedulers.io thread, this works similar to livedata as when a change is presented to the database it automatically fires and updates the UI. An example of how this is used can be found in the DebugFragment towards the end of the onCreateView function.

# Chat GPT

This is using the network module to do most of the service call, the only difference for this api call is the prep work. This can be found in the GptHelper class, here I wrap the question in the proper syntax as well as read the answer from the response so I can easily display it on the view. As chat gpt evolves you can add more parameters to your queries as you feel fit.

# Admob Ads

Admob is pre installed and comes ready to use (assuming you replaced the keys). To use the interstitial ads first you need to initialize the service, I do this in the onCreate of the MainActivity. Then the ad needs to load then it could be fired, I do both of these in the fireOffAd function on the MainActivity. So all you need to do is call that function and wala an ad should be loaded. DebugFragment has this already in place to test.

## Billing

The app even comes with the Google billing 6.0 wired up, this does need more setup on the web console but is pre-wired like the rest of AppKit. Once you have created a product you will need to add it to the app, this could be done in the IAP\_Helper class. Here you will find a function called queryForSkus, in there are a couple examples. Add the sku to either a INAPP or SUBS productId, currently it says NA. Similar to Admob this needs to initialized in the MainActivity as well (on line 78 in onCreate it is done). Once you have the above implemented you can easily test this in the PurchaseFragment, you can get here from the DebugFragment. If you have successfully implemented everything you will see a list of items to purchase on this screen. Pressing one will launch the in app purchase dialogs.

# Google Cloud Messaging (Push Notifications)

Most of this is setup behind the scenes, once implemented change the default\_notification\_channel\_id to your app name and test it out. You will need to grant permission to receive notifications, the code to do this is in the PushNofiticationController. An example of how to use this could be found in the DebugFragment under the function testGCM().

# Permissions

I have included a class called PermissionCheck that handles the logic for asking for permissions on Android. Currently Location is used as an example. With location there is a slight difference as we can get one of two kinds so this code is in the MainActivity but the other general permissions could use the requestPermission function in this class. An example of the Location is in the DebugFragment as well as an example of a general permission (camera). Do not forget to add the permission to the manifest!

# Location

Location is fairly simple, although I am using the newer method recommended in the developer documents. This allows us to know if the user has accepted the course or fine location permission. An example of this usage is in the DebugFragment in the function testLocation(). Here I check if the user has the permission or not, if not I request it. You will need to request the permission again as I did not implement a callback for this, I would listen to the onResume with a flag to request permission if that is the reason the fragment was paused.

# Intro Activity

Just like big name companies, you can have an introduction section that is only shown once too. The app has one that even requires a user to sign in so you can know who is a new user vs returning user, this is completely optional. These fragments can be gutted and replaced with anything you like, I just liked the code so I figured I’d share it. To activate the Intro section place the intent-filter from the MainActivity to the IntroActivity. Now the intro will run on start up if the user has not seen it before. The intro has a custom fade in/out animation that can be used anywhere in the application. A simple ensure the user has checked before continuing as well as other little tricks I used in my last app that I was to lazy to remove from my AppKit. I added a skip button to bypass the sign in screens.

# Selection Bottom Sheet

I have made some nice rounded bottom sheets that are found in the settings package, but I have also added some custom selectable bottom sheets. This is my answer to the spinner that never got UI love. You pass this bottom sheet a list of custom models found in the BottomSheetHelper, this class also has the converter that creates this list in the createBottomSheetList() function. You could make this a single select list or a multi select list, the result is passed back using an interface. An example of this is found in the DebugFragment.

# CustomDialogs

There are two parts in this package, on the one there is a custom dialog fragment you can turn into your own custom dialog, this requires a CustomDialogModel to be passed in. Then there is the CustomDialogs class that contains the wrapper for your quick alerts. There are some snackbars, depending on what you want and there is a single toast in case it is still needed. There is also a dialog using A. Follestad’s library, I love this library and have been using it for years. Star to show your love and so it continues to be awesome. https://github.com/afollestad/material-dialogs

# Settings

I developed this style around the time Samsung added it to their phones, yes it’s that old so feel free to dispose of the design. I just love the parallax feature when the user scrolls so, I keep it. There are some goodies baked into the settings like two bottomsheets for the about something screen, an example of the Utilz features like send feedback or share app, and rate app using Google approved method. All these features are located in the Utilz classes and can be triggered from anywhere.

# AssetLoader

This is useful for when you have a complicated set of default data that you want loaded into the app, there are many ways of doing this but having a single json file makes maintaining the data easy. This class will load that file for you as if it was a network call, one can even make this a back up to a network call.

# Utilz

Over the years I have collected a ton of misc functions that do cool stuff, I have added them inside the Utilz package which is in the core module. In here there is an Account Helper that is used to launch the Google sign in screens, here you can sign in, sign out, and delete an account. A ModelConverter that helps map network response models with database models, this comes in handy when you want to modulate everything! If adding network and database items, use this class to map them to one another. StoredPref, one of my oldest but most faithful lines of code. This easily allows you to read and write to the stored preferences and keeps everything nice and neat, just add a key up top and its functions below copying the format. The Utilz class has the rateApp and open in browser functions used in the app.

# UtilzDateHelper

Dates in programming are always so much fun to work with, so I have built a way to make it a little easier to digest. The functions are commented but you could build a date in millis or in human readable form, you could then convert it to either form using these functions.

# UtilzRevealViewAnimator

This is neat piece of code, I built it years ago and it still looks amazing. It does require newer OS’s (8.1+ I believe), to reveal a view make sure it is invisible and then run the revealView passing in the values (view.width/2, view.height/2 for center point).

# UtilzViewHelper

Here lies the function that fades in the view in the IntroSection, I have also added the code to show/hide the soft keyboard from any view, as well as a function to request focus with the keyboard on a specific view... Helps in the event the user types into emptiness after the keyboard is magically shown.

# ErrorController

This class is the wrapper/mapper for all logs, errors, crashes, and exceptions. By having everything funnel into one class you can come back later and connect them to your analytics of your choice easily. You can also add a custom string to make finding the logs easier.

# Dark Theme

There are several different ways to theme an app, but this app comes with a simple solution for this. By connecting all colors to either a dark or light colors file the app will load the appropriate color based off the theme. One more complicated views you can add a night mode qualifier that is set to night, then in dark mode this will be loaded instead of the light themed styles giving you more flexibility of styling your app appropriately.