Write Up

1- After going through the requirements and asked components for the project, if I was granted more time, **first thing** to do is to add more usability features to the app, such as a search bar for example, or filters where the user can have more of the right to specify what kind of articles he/she is looking for, another feature is to add a login feature from where the user can have subscription option , where he/she can subscribe to his/her favorite category and get notified whenever there's a certain event for his subscribed category through push notifications, these are all features that could make the app more usable and function better, as part of the UI I would have preferred to inject more animations to the app, these things make the app have a better appeal to the user, as in **priority** I see that first thing is that the user should always be allowed to be part of the app, by registering and logging in, interacting probably commenting on articles, this makes the app daily used and would be a very component part to add, second thing is the filter part and the search, where it would also help the user to be more **on point**, last part is the animations and effects part where it's the icing on the cake which is always a nice part to have in the app and provides the UI joy to the user.

2- Asynchronous tasks mainly give you the privilege that you don't have to wait for the response of the service call, the executer continues to function and is allowed to open other threads in parallel while the background thread is executing it's calls and then when the call responds, it gets the executor back to continue the onResponse actions, which in our case in **Bolts** is the continue with, on the other hand the callbacks, and the normal implementation of volley makes you have to wait to the service call responds, the extra road given by the Asynchronous task isn't applied here and the app's executer has to hold till the service call responds with either a success or a failure for the executer to be able to open other threads, it's only allowed one road with threads in traffic.

In this project I made 2 versions, **first version** was with using bolts, and the second version was by using normal volley with callbacks, they aren't interchangeable as their methods of implementations are different, I integrated bolts with a library called **Volley tickle,** which is a library mainly derived by **Volley**, the big benefit of Volley tickle over Volley is that **it can execute the request without having to open it's own background thread,** unlike volley which by default opens a background thread whenever given a request, in

our case Volley tickle is more convenient because Bolts already opens a background thread through instantiating the Asynchronous task, so it would be irrelevant to open a thread when there is an open when at the first place by bolts.

The second version is done regularly with volley and callbacks, whenever the service call responds the it reflects in the callback and the normal lifecycle of the app continues.

Using Asynchronous tasks is very necessary in a case where the service call's response retrieval isn't a crucial part for the app, that it can continue regularly with calling other tasks till the task gets a response where it gives the app more freedom to function without being dependent, using standard callbacks is more useful in the case that it's very necessary to continue the flow of the app, where the response of the service call is crucial, that the executor can't call other threads and has to wait for the call's response, Scenarios are for example if I'm calling 3 lists from different service calls with different object responses, here asynchronous is better and provides better performance, on the other hand normal callbacks are better if for example I'm doing a submission for a form or an

article, that needs to respond with a certain response for the app to go and handle other tasks, here callbacks are better in use.

The project was implemented through the MVP design patter where the data in the model are separated from the view, and presenters with there interfaces act as a middle layer between them in order to present the data to the view from the model, I decided to do interface implementations as they come very useful in testing.