**Progressive Mobile Application(draft)**

(PMA, an ever responsible syncable mobile-app concept)

v0.1

By Xinyue Zhao

*- “A small leak will sink a great ship.” by Benjamin Franklin*

*- Great oaks from little acorns grow.*

**App should provide information continuously**

There’re a lot of development methods or patterns to make our mobile-apps robust. Because almost all mobile-apps have same task which means to provide useful information to fill the satisfaction of users. Users download our applications in order to have information they’d see.

**Problem: App runs with network well, however...**

Since begin of Internet we have to encourage getting information remotely. Today we are used to live with cable ports at home, mobile devices underway, wifi connecting as you visiting your friends. All telecom hardware, including software, which we’ve invented and improved only have some piece of purposes like better communication between people, shorten and accelerate the speed to help people get information.

Information, we call the data. User must see data. There’re a lot of environments under where the mobile-apps could run. General speaking we launch an app only with internet and the app can only run and be connecting to fetch data using API through network. With excellent status of network the app looks health and handful. Amazing idea which is driven and leading connective status is remarkable like a battleship cruising over peace of ocean, no storm, no tornado. However, the reality is cruel, no one can guarantee that one day the captain has sufficed current status and would like to surf in a wrong direction. The network can be broken down round you. You might have LTE or 4G but you have reached your top volume, you might be camping without wifi but your mobile telecommunications signal shows full-status on your device, we call this [offline-like][[1]](#footnote-0). **If we are regardless of the network state, the mobile-apps don’t work stable and consistently.**

**Problem: In source-code you see too many data-source and Different ways to request data.**

A mobile-app needs network, a mobile-app needs data-source. Macroscopically speaking a mobile-app is a represent of all aspects of data. Messy organization of data results the app into a situation of maintain accident and difficult of upgrade, it looks like this room:

(screen-1)

What we want to see is this room:

(screen-2)

**Solution, the PMA**

This article proposes a design point of view how we can use the concept of [Progressive Mobile Application][[2]](#footnote-1) to minimize the loss and boring behaviors of the mobile-app. This concept is independent of the different implementation details, i.e [architecture][[3]](#footnote-2), framework, platform etc. Hope to get some empirical consensus from readers.

**Where does PMA come from?**

A basic concept of PMA tries to achieve the purpose that a mobile-app provides uninterrupted information or data, which means the users won’t lose sustainable state of the mobile-app. The original idea has been introduced by Google.Inc with the name [progressive-web-apps][[4]](#footnote-3) where the Googlers focus on [three core features][[5]](#footnote-4) of a modern web-app.

[

* Reliable - Load instantly and never show the downasaur, even in uncertain network conditions.
* Fast - Respond quickly to user interactions with silky smooth animations and no janky scrolling.
* Engaging - Feel like a natural app on the device, with an immersive user experience.

][[6]](#footnote-5)

The PMA emerged from the foundational of progressive-web-apps and reforms this concept on common issues of mobile-app.

Different from web-app, a classical [mobile-app][[7]](#footnote-6) provides some natural features or facilities which browser-based-app can’t.

* Full-Control over available APIs of operating-system, i.e camera, file-system, network etc.
* Supports local native database like SQLite.
* Direkt graphic enhancement from hardware like GPU.

The PMA takes advantage of all these hardware and software to give users responsible feelings with mobile-apps.

**Foundation of PMA**

* **Database**. We need storage to save and cache data. In order to avoid losing data after application process being killed, a real long time persistence is necessary.
* **Data flow**. We need a [single & unitary] processing to finish transferring data from internet to UI. With database we will cache all data in and load them, populate them on the display of device.
* **Algorithm**. We need an algorithm to process selection of data when the response of APIs has arrived. To define how the mobile-app will save data in storage, when the mobile-app should load data from storage. The behavior of error-handlings of network or database must be considered.
* **Network context sensitive**. From UI to Non-UI parts of the mobile-app must be sensitive to network connectivity. A gracefully continuously responsible feedback about connection-status helps us improve program codes and application’s stability under different net-situation.
* **More payload-data**. When a new mobile-app launches on your device first time, the app should show some built-in data in order to avoid wide range of progress-bars and blank areas. It could make users boring if the mobile-app will be downloaded and used underway without wifi, but only cell-net.
* **More chances for reload-data**. The mobile-app must give users to fetch new data from backend in order to refresh local repository. The conditions of reload should be vary enough, i.e network switch from 4G(LTE) to wifi, [switching from the connection-status to airplane-mode][[8]](#footnote-7).
* **[Just Request][[9]](#footnote-8)**. The mobile-app sends mega http-requests to backend in order to get data or interact with different clients, but they are tasks synchronously. A PMA should request server without wait responses lengthily , endlessly and asynchronously. Build charm UI-Elements to confirm each pair (request-response) to pretend a smooth UX. Save every request that has already been out, ensure yourself that no response will be lost, [deprecate requests after all responses have been handled][[10]](#footnote-9).
* **Search, everywhere.** The PMA will peek up all data from local, the search performance is obviously more efficient than the apps which only know online-data. Try to build a mobile-app which provides **search-button ever on top-right of the app-bar.** The search-button works with local data which can also [navigate][[11]](#footnote-10) the users to anywhere inside the domain of the application. Provide “search more” as fallback asking the backend, sync them to the local database and display the fresh data. The users use our application in order to see data and information, dare to explore and ask questions is a potential usage behavior as well..
  + The search feature can improve the richness of local database.
  + The search can give the backend more chance to collect user **special** taste.
  + Promote functionality of backend in some areas like data mining, the machine-learning. Improve API’s efficiency and accuracy.

(screen-3)

* **Think about “sync” instead Save, Update, Delete actions**. Don’t forget to make a copy of data which was saved or updated in the backend successfully. Furthermore remove data after the app deleting data.   
  Parallel to backend and track possible errors from it, don’t sync when any error happened.   
  Use **[Just Request]** to save http-request if it is needed so that the mobile-app won’t forget what the user has done.
* **Let backend know status of device.** When the mobile-app requests on backend try to include necessary device information like screen size, RAM etc. For some cases the backend can reponse client base on needs. Don’t forget classical “out-of-memory” exception that slapped us in a lot projects.
* **Make application features straightforward.** Keep users understand. The official [Material Design][[12]](#footnote-11) is pretty cool, we should keep it and follow and comply, however, here I point out some which are very important but always forgotten.
  + According to the icon of button to identify the function. There’s a suit of standard [icons][[13]](#footnote-12) provided by Google Inc. which we can use in our projects directly. The advantage of using them is that a lot Google Apps or material-based apps use them as well. As the user's experience grows we don’t have to guide them how, what, the button would command after clicking. Don’t be **too demanding beautiful**.
  + Avoid **unnecessary** popup or question conversation. Not all users are patient enough to wait for your question sequence with some selections or decisions. Disperse the user's attention from be caught in shame of thinking. **Using app with fun is important**.
* **Flow and Navigation**. The mobile-app has navigation to let users be navigated at correct feature places. The menu is also an important part of application. Some apps might have pagers on which users can swipe with single hand.   
    
  Don’t do navigation in conflicting ways:
  + Keep menu in clear classification. Avoid wrong placement of subclasses and parent classes.
  + Design navigation with clear information. If a bottom-bar is provided add label under each tab. If navigation-drawer is provided don’t make labels too long to the right of icons. If tabs plus pagers (screen-4)don’t use icons.
  + Let app know minimum data to show after being navigated. [**More payload-data**] supports a flow navigation and doesn’t block user from one place to another.

**How does PMA work?**

To run PMA we need architectures. As mentioned before, this concept doesn’t focus on how we implement them.

The principal of PMA is about caching data on device and the mobile-app will show data from local firstly and try to load fresh information from backend with handling different UX. Network-Status will be handled under control of confirmation from users. There isn’t different UX between offline or online.

* A good choice of pattern of software-development is important. Here are two typical patterns:
* Model–view–presenter + Repository-Pattern
* Model–view–viewmodel + Repository-Pattern
* Repository:
* Firebase(fire-store)
* Local database + Algorithm

**Database**

* **Local DB(SQLite, Realm etc).** Use local database like SQLite or Realm the mobile-app must control the logical or relation between remote-data and local-data. Use different an algorithm mentioned in part of foundation before to minimize consumption over network transaction. We call this algorithm **the select**:
  + The select returns local-data ever. [The PMA shouldn’t push back remote-data directly][[14]](#footnote-13). The PMA represents the data that have already been “filtered out”.
  + Use traditional AI functionality like [Breadth-first search][[15]](#footnote-14), [Depth-first search][[16]](#footnote-15) to organize data saving and remote fetching. Eventually the select will push back data from database.
  + Use [Neural Networks API][[17]](#footnote-16). This topic might be beyond of our PMA topic, however, here I have to point out some reasons why we should define the select with NNA.
    - The select should be flexible. The traditional search AI has fixed route and logical. If we want to extend or improve features like performance we might re-code or engage more times. Use frameworks which base on NNA i.e Tensorflow Lite, let the device learn and improve itself, how to select data and save data.
* **Firebase.** Since Firebase came to market it has been the fate of mobile-apps to transfer.

**UX, avoid blocking, keep smooth**

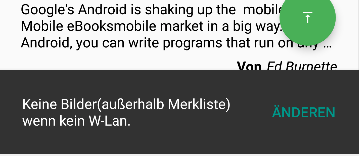
* **Refresh bubble.** The PMA shows data from local, it needs some background requests to fetch fresh information. Don’t let application block our smooth user-feelings. Show a classical bubble like Facebook or Tweet to notify your users the new coming signals.

(screen-4)

* **Notification.** The PMA knows every request(Just Request). Compare with other apps which don’t mark sent requests, the PMA can notify user every response according to each request. Build notifications for every response if possible.
  + Is the mobile-app foreground, show some UIs like snackbar or just stick bubble(Refresh bubble).
  + Is the mobile-app background, notify through build-in notification manager.
* **No Toast, less Dialog.** Do not try all messages or information with popping up a dialog. Too many dialogs would impede users at some scenes where they must do straight forward, here some tips, we won’t need dialog:
  + Registration-scenes, like user-data, product-data etc.
  + Scenes at where we’ve already used some build-in dialog like time-picker, date-picker. Try to notify users with snackbar or just a normal view on layout.
  + Interaction-scenes, like chat-session, video-session, **while** shopping-session with shopping cart etc.

Don’t use toast. It brings limited information and interactions. What dangerous of a toast is that the toast could transfer user’s eye-target. Think about some errors happened in scene-A and >= 1 toasts would be shown for feedbacks of these error-handlings when the user has been at other scenes or even 3rd mobile-apps. The toast can’t be controlled easily within the feature-frame.

* **More Snackbar.** The snack-bar is a modern design element which is introduced by Google.Inc [Material Design][[18]](#footnote-17). The biggest advantage of a snackbar is its availability of interaction. Try to build button on your snackbars and give them behaviors.

 (screen-5)

**Checklist of PMA**

A PMA should have these features:

* Avoid having Large area blank **plus** endless progress bar.
* Avoiding showing empty-state or no-data messages.
* Give users more chances to reload data when the connection-status of device has been changed. Don’t let them only do with Pull-To-Refresh.
* Show errors from network as soon as possible and provide reload chance.
* Keep offline and online showing parallelism data, avoid a huge different UX between these, keep users being no-feeling if the mobile-app is running in unstable network or environment.
* Give every App-Update some Payload-Data. After installation the mobile-app should show data as soon as possible to win user's heart for next version.
* The product must be stable, clear, correct handling, transparent circumstances, maintainable. The developers can find where a bug might come from. Any upgrade or bug-fixing handle events or messages in the app correctly. The codes should have transparent “come from” and “go to”. The stable running of application is remarkable.
* Have you see the search-button on top-right of app-bar.
* You understand what the UI works at first glance.

**What kind of mobile-apps need PMA?[[19]](#footnote-18)**

These kind of apps are for PMA perfect.

* **News**. Because major users read news underway like in subway or bus. A weak network is a common problem to this case. A relative long cache of news or all articles of some rubrics might be good UX for readers. Users can use Pull-To-Load or load-button to get new data. Avoid requesting and asking backend for peep new data. **If the user returns back to wifi the app must give significant prompt to ask pulling down newest information as soon as possible.**
* **Fashion**. Keep large amount of media data like photos, gifs, small videos in device, cache more and more product data , these bring a fast and affinity UX to users and help us create better animation locally. The data could be updated by clicking some charm UI-Elements after the app peeping backend in [background][[20]](#footnote-19) for new data.
* **Social-Network(SN)**. Like News-App above people would use SN-App very mobilely underway. Sometimes the SN-App plays role like a news-service, users can post comments, write messages on others etc, they don’t want to wait feedbacks just “post and leave”. Think about “Just Request”.
* **Instant-Message(IM)**. Rich interaction is core of IM, users won’t be blocked by different requests include chat, emoji, sounds, small videos. Think about “Just Request”, don’t delay our users on screen.
* **Customer Relationship Management (CRM)**. The core idea of PMA is that the data should be cached and saved locally. For cases like CRM which are the mobile-apps always depending on massive data need long-term saved data and refresh on-demanding.

**What kind of mobile-apps should avoid PMA?**

I don’t think there is any limitation over PMA, however, I have to mention two points from my career experience:

* **Real-Time.** If the application needs data against real-time i.e stock, currency, traffic etc. The users of such applications are often the groups requiring exact data. Because PMA has the purpose is to keep users seeing／using／operating data without minds of network status, they could be the data 2 minutes older, or even worse, it’d bring economic and interest losses when these groups of user use such data or statistics, thus, it is dangerous.
* **Limited by law.** The mobile-apps would be launched in different countries, every land and territory might have special proposed law(bill) about privacy and data protection. The PMA must be limited in range of authorized law area. Also some your clients don’t what their data be cached in every devices which have their apps, thus, you should   
  plan and prepare whether you need PMA.

**Ending**

The PMA is a concept which stays at a point of view to solve problem such as how the mobile-app would provide continuous data being independent from network environment. To improve application’s sensitivity of device according to the surrounding status the mobile-apps need to be intelligent underway and show different friendly UI to tell user what happened, what will be issues.

The PMA is always “local-first”, which means the local data which have been stored through every request／response are priority to use. In order to keep users being with “non-offline” feelings the PMA must provide different ways to fresh data and save them in local database, meanwhile, show different friendly UI like before. How to save data and use them to render on display depends on different algorithm, **the select()**.

Try best to let the mobile-app be available and usable under weak network condition. This is a good appraisal of your PMA. Don’t show too many empty views and long loading progress-bars, they are almost killers of a good rated app.

**Keep your apps running and don’t convert them to PMA if your current concept or architecture works.**

“avoid empty, short loading”

1. An offline-like is described here not a situation that the network is off completely or no internet,

   however, it could be network-connected scenes with deception. [↑](#footnote-ref-0)
2. I use PMA at rest parts of the article. [↑](#footnote-ref-1)
3. Actually a lot of architectures have already introduced clean structure to help us make PMA completely. [↑](#footnote-ref-2)
4. Visit here <http://tinyurl.com/z2lj2pc> [↑](#footnote-ref-3)
5. I’ll introduce same 3 derivative cores from these Google threes. [↑](#footnote-ref-4)
6. Visit here <http://tinyurl.com/z2lj2pc> [↑](#footnote-ref-5)
7. In this article the mobile-app is limited in Android or iOS. [↑](#footnote-ref-6)
8. Different mobile-os provides different mechanisms to handle status changing. [↑](#footnote-ref-7)
9. Think about: Do you want to wait http-response after sending request or just continue to next step? [↑](#footnote-ref-8)
10. Including responses(requests) already expired. [↑](#footnote-ref-9)
11. Define a json file which contains key-value i.e “home”: “intent://com.app.open.home”. [↑](#footnote-ref-10)
12. https://material.io/guidelines/material-design/introduction.html#introduction-goals [↑](#footnote-ref-11)
13. https://material.io/ [↑](#footnote-ref-12)
14. There’re some exception cases, I will mention that in later lines. [↑](#footnote-ref-13)
15. https://en.wikipedia.org/wiki/Breadth-first\_search [↑](#footnote-ref-14)
16. https://en.wikipedia.org/wiki/Depth-first\_search [↑](#footnote-ref-15)
17. https://developer.android.com/ndk/guides/neuralnetworks/index.html [↑](#footnote-ref-16)
18. <https://material.io/> [↑](#footnote-ref-17)
19. No limit actually, some apps are really recommend [↑](#footnote-ref-18)
20. Generally the mobile-app should ask backend in order to know whether some new data should be loaded, at end the confirmation will be represented on UI, the user should decide whether the data could be loaded or not. [↑](#footnote-ref-19)