Design Summary

The Meeting Scheduler is designed to help users manage their meetings agenda in an easier way. So designing the App as simple,useful and reliable is crucial for the development. As high quality designs are always closely dependant on good design models, patterns and principles, here is a brief summary for the design of the project.

Design style and principles

As a helper application, the Meeting Scheduler is designed to be able to used in easy and simple way, so the following design styles are followed:

- **Simple curves and geometries :** To providing a simple impression and using experience, the content and component boundaries should be decorated with simple curves, and the background can be rounded if necessary.
- **Multi-color and font contrast for better readability:** The interface should be decorated by adequate colors to convey a sense of design and avoid aesthetic fatigues. The high contrasts between several colors can also help to improve the readability of contents.
- **Functional interactions and animations:** To have a better user experience, the app decides to use large amount of fragments to replace activities in order to generate good functional interactions with users’ action, also the transmit between typical activity such as Main and Search activity should decorated with soft animation.
- **Information data visualization:** A meeting schedule timetable is provided at very convenient gesture position(left first in bottom navigation bar) since we believe a visualized data is always better than any words.

Besides, two important design principles are obeyed during the whole design stage. They are Single Responsibility Principle(SRP) and Dependency Injection(DI). The SRP is strictly present in every single activity, fragment and java supportive class. Each one class only take one responsible function, they collaboratively work together to provide services to in the app. The second principle DI is applied for the data management for MeetingModels(The Meeting object). To better schedule and manage the meetings data within different activity and fragments, the data has only one base version which is in the homepage meeting ListView. All others requirement for data such as search and display will be injected from homepage meeting ListView data content.

Design patterns and model

To provide high quality of delivery, the project is designed and developed based on several design patterns. Typically, we choose to deploy the factory pattern for login view model. The database and helper classes are accessed under singleton pattern. For the register, login and feedback modules, we apply observer pattern to better coordinate the delivery of messages. The adapter pattern is also used in the scrolled meeting list view as well as the feedback part. And finally the strategy pattern is used to fit the varies requirement for meeting models data.

As for design model, most of the modules in the project are based on traditional MVC model to manage the model view and controllers. We used innovative MVVC model in feedback part to reach a high level in performance and expandability. We decide to store the data both locally by SQLite and in online server based on Google Firebase to better fit the using cases of variety of data.

Innovative technical design

There are several innovative technical designs in the project based on latest techniques. In register/login module, we integrate Eventbus, Lifecircle&Livedata and ViewModel to implement the functions, MVVM model and RecycleView are also deployed in feedback module.

The Eventbus making our functions do not depend on context, and it become unnecessary to inject context during scheduling, which significantly improves scheduling flexibility and removes the coupling brought by handlers. For high-priority events such as login status, it is possible to create a well-structured and closely-organized notification system.

The Lifecircle helps reduce our code coupling, realizes automatic monitoring for the data, do not need to manually or proactively call in several fragments. This not only reduces the code amount but also increases the degree of maintainability. The Livedata is to make sure the called back only happened when the component is activated, thereby refreshing the corresponding UI. It brings significant convenience and robustness for the project since the number of fragments based on main activity are high, which means the context can be switched frequently.

The ViewMode from MVVM mode in both two modules allows our activities and fragments do not need to take the responsibility of saving data in themselves and are handed to ViewModel, which greatly reduces the decoupling of data and views.

The RecycleView provides us actually operating ViewHolder as well as animation in feedback interface, with high performance and good expandability to other modules.