Introduction

As explained in the specification of our application that we handed previously (that I suggest you to have a look at again), the goal of this application is to provide a way for students of Efrei Paris to propose ideas to improve the school.

Despite our great efforts commited in this projet, we did not manage to implement everything that we wanted. We realized that there are tons of things to deal with in Android development and that any detail takes a lot of time to be implemented.

If you compare our current application with the specifications, you'll notice that the filtering functionality for displaying propositions is missing, as well as the neutral vote button. Please read more about our ambition for the evolution of our application in the section "upcoming features" further below.

Feel free to try our application, start by creating an account. Then you can see a sample of propositions that we generated. Try to create your own propositions, vote to your own or to other's propositions.

That might not sound like much, but keep in mind that everything you do is synced with a remote server and that what you do will appear on someone else's phone. Don't hesitate to try the application on two different phones at the same time. (Although data are only fetch from server when you open the app for now. Again further explaination in the "upcoming features" section below).

Feel free to have a look on our server back-end code on this repository: https://github.com/Tomro8/DB_Mobile.

Find the Android source code on this repository: https://github.com/Tomro8/Revendiquons. (Branch master is the production branch).

Now for the most interesting part, I invite you to read about the architecture, technologies and libraries that we used for the project just below.

Technologies

Backend

Revendiquons' data such as user information, proposition information or user votes are stored on a remote **MySQL** database. We interact with the database using **PHP**. Our web server is hosted on **Digital Ocean** (a cloud hosting provider).

API Calls

API calls are performed using the **Volley** library as recommanded by Google for HTTP requests. It does the same job as HttpUrlConnection with the benefit (among many) of handling the background execution on the developer behalf. This feature is very convenient.

Persistance

Once data retrieved from the server, they are stored on a local database on the mobile itself. We use a **Room Database** which is now Google standard for persisting data locally. Room brings an abstraction layer over SQLite, allowing to interact with the database in more seamless way. The goal of persisting data locally is to allow the user to use the mobile application even when he does not have network access.

Architectural Patterns

This application uses two major architectural patterns to shape the application design: the **repository pattern** and Google's **MVVM** pattern which stands for **Model-View-ViewModel**.

- The Repository Pattern's purpose is to provide a clear a seamless interface to the database. It's an abstraction layer over the database acting as the *single source of truth*.
- The MVVM pattern has greatly shaped the design of our application. In fact, it is the core concept around which the other components of the application are revolving. It brings several key features. First, The data related to an activity are stored on its assigned ViewModel class. Therefore, when the activity is destroyed, all the data are still right at hand when the activity is restarted. This way, data are displayed quickly, therefore improving the user experience.
  Second, MVVM uses **LiveData** objects to observe data changes in Room database. Those Livedata objects are themselve observed by the UI. Therefore, each time there is a change in the database, the UI is instantly refreshed to display what's in the database. Lifecycle issues are handled by the LiveData library and we do not have to manually hydrate the UI.

Expandable Recycler View

We wanted to display propositions in a way that would allow the user to quickly understand what it deals about. In the mean time we wanted it to be possible to learn more about a given proposition by staying on the same activity. So we decided that a list displaying proposition titles with a description unfolding underneath at user click would be the perfect UI component.
Recycler View (which is the new Google standard for list rendering over old Listview) did not support expandables items. Hopefully, we found a quite popular library on Github called **Expandable-Recycler-View**.
https://github.com/thoughtbot/expandable-recycler-view
It is bases on Google's recycler view and add this expandable on click functionality.

Upcoming Features

An application's development is never complete. There are always things to improve or functionalities to add. Revendiquons is only a newborn, and the potential upgrades it needs to become a real application are almost infinite. Nevertheless, the nextcoming features we want to add to this app are:

- A toolbar with a logout button
- When the user is logged in the application, we he relaunch it, he is automatically logged in and redirected to the main page.
- A refresh function: when the user scroll's up the list, data are fetched from server.
- Data fetched from server periodically for a Real-time application.
- Animations: when going from one activity to the other, loading snippet when fetching data...
- Filters to select what to display in the list.
- Email verification at account creation

Conclusion

This project has been my favorite of all projects I had to do at Efrei Paris. Because it required knowledge from different fields such as databases, API, servers, PHP, Android and also front-end design. We tried as much as we could to stick to best practices and to Google's standards for Android applications. We had a lot of fun exploring various technologies in order to find what suited our application best. As for me, I truly discovered a field that I am passionate about and I look forward to do more Android development in the future.