Wunderlist Technical Challenge

Author: Alfredo Cerezo Luna.

Document scope:

The purpose of this document is to facilitate some technical decision and clarifications.

Project description:

The application consist of an list of task defined by the user, the requirements are:

- The user will be able to Create an undetermined number of tasks and they will be showed in a list.
- Each task will be stored in a SQLite database.
- The user will be able to check any of these tasks as completed, which will be determined by a check icon on the left of the tasks.
- The task will not be deleted from the list neither from the database in case of being checked, this is a design decision, but technically is totally achievable.
- The user will be able to remove any task by swiping it out of the screen, the task will be removed from the database too.
- Reorder any task by long pressing over the element a performing a drag and drop gesture.

Methodology adopted:

Not following TDD, TDD is a suitable methodology when the project is intended to be alive and maintained for a long period of time, and also to preserve the project robustness during the development process against any change/ refactor in the code, since the scope of this project is to develop a demo, and honestly, I don't think you are going to implement my list in any future Wunderlist project, I decided to go against what my resume states and to sacrifice the tests in favour of having a decent software architecture and code.

In case of designing a Test plan, it should be divided in:

- Unit testing based on jUnit and Java pure classes.
- Instrumental test, to test any Android provided functionalities.

App architecture:

The application follows an Uncle Bob's Clean Architecture approach, dividing the application in layers ensuring frameworks/libraries decoupling, making it easy to maintain and to scale, and easy to implement tests with mock modules in a future.

Third parties libraries:

- **Dagger2** for Dependency Injection, this will make really easier to implement future test plan or to change any layer of the architecture.

- **DBFlow**, an ORM database library, to maintain and deal with the SQLite database. Since my experience with Data bases is not as extensive as I would like, I took the decision to implement its layer using an external library, but in a future, thanks to the subdivision of the architecture in independent layers, if a I have a ninja expert team mate in databases, he would be able provide a new module which will be very easy to integrate in the software using Dependency Injection (Dagger2).

- **Apache commons**, to generate a unique ID per task.

Network layer implementation:

Each layer is decoupled from each other, and they only communicates between them using an interface, this architecture includes the repository layers from where the data acquisition is performed (Task Gateway), but they also are complete asynchronous from each other, so each Use Case execution is performed in a separate thread (CommandExecutor, command design pattern), so having different data sources is totally achievable.

The data integrity in the View is guaranteed since the data structure containing the information being displayed in the list is synchronized.

All the process described above is for user update petitions, in case of server update petitions, **a push mechanism** must be implemented, for example **GCM** (Google Cloud Message, and its super powerful brand new Security Token API which is awesome providing a unique ID per application installation).

Once the mobile phone receives the push notification, it starts a new Use Case where it ask the Network layer for a specific task indicated by the push message or maybe a bunch of tasks ready to be downloaded instead of only one (performance and data costs improvements).