# [GSoC'21]Irdest Android Client-Blog II

Ayush Shrivastava

Sunday 11th July 2021

## Prelude

Hello! Good to see you here :) This blog is mostly a summary of work done till now under the first coding phase of summer of code of '21. Picking from the end of *previous blog post*, we planned implementing chat feature in the application module, but due to the aforementioned massive refactor in the entire codebase and upgradation of existing modules to support modern hardware the chat API has been deprecated, and some components, because of not being part of CI, got broken :( To implement features in the module the very first step was to get the project build properly, previously (maybe) due to migration from different version control system to GitLab and that massive refactor there were some unidentified problems that did not let the application codebase build properly, also the main lead of the picture `android-support` crate, not being a part of our GitLab CI workspace too, was broken. We fixed all this entire stuff in multiple different steps, each solving a mini problem and writing CI for each missing component so that we or anyone joining the project never encounter similar problem(s) in the future.

## I. Fixing the Android Application Codebase

The application codebase was considerably broken and for the **very first time** when I built the application, it **instantly** said *build failed in 10ms*, which is really very weird as when for the very first time you build an android application, it takes noticeable time(∼6 minutes), this time is for fetching the dependencies that are declared in the dependency management file of android codebase(the ones with the `build.gradle` name) followed by compiling the android project codebase. It was quite astonishing at first sight, but on closer look to the files present in the android codebase the cause was observable. It was the presence of dependency archives in the android codebase and their corresponding XML files too, and since these dependencies were already present, studio didn't take the pain of fetching them from maven. So the question arises, *When the dependencies were already present still then application didn't compile, why?* Actually what happens is that these dependencies' XML files are editable so even the slightest edit in them makes them useless and studio doesn't even report any kind of problem with them, another thing that happens under the hood is that studio stores these dependencies in its local cache so that when user recompiles the application, no time is taken in fetching the dependencies and it can peform the *real build.* Also, by time these caches get corrupted and usage of *very* old cache does not let the project work in the way it should.

Okay, now let's come to the point how we fixed it. As by now it should've been clear that the problem was the existence of binaries and XMLs of dependencies present in the android codebase, so the solution that I anticipated was the deletion of these files. It was *not* sufficient. After doing this dependency management did go as expected, but still the build failed :( After some more inspection, I found there was some problem with `gradle` executable scripts as well and the `gradle-wrapper.properties` too. So I just referred these scripts from my previous working projects and it finally started working, a moment of joy, after many days + nights of pain ;) As this problem was fixed, after working on some other crucial matter(see next section, **II** one), we wrote a CI pipeline specially for our android application codebase so that it doesn't break again ever in the mainstream. The android-application pipeline comprises of the 3 stages, in which its lint is checked, followed by build and then the tests are run. In the upcoming coding phase we plan to make this CI pipeline even more robust and enforce stricter formatting rules, introduce Static Analysis and run android Integration Tests on GitLab CI, well we'll discuss it the next time we meet, leaving some topics for then :eyes:

You can find the corresponding MRs here:

- Fixing the android application codebase: *[we/irdest/!21]*
- CI for android application codebase: *[we/irdest/!23]*

## II. Fixing the FFI Layer

After the previously mentioned refactor, everything was working fine, only android specific components of the codebase were broken. A part of which, was our FFI layer, the `android-support` crate. This layer still held references of several deleted and deprecated APIs, therefore compiling this crate too gave a bunch of errors. Fixing them took much more time as this crate was written in Rust and then I was not that fluent with it. So fixing it included updating/modifying existing functions or we had to remove functions as well because of the deprecations. A nice challenge that we encountered was saving the state across multiple platforms(supported hardware), because the crate we used for saving state, provided support for almost all the operating systems other than Android. So what we did was using the knowledge of android that an application has access to its own private storage which no other application/service can see, so all we needed to do now was to find this directory in android device file-system, this path we achieved using the ADB, now we investigated where our crate went wrong, so for this we dived deep into the crate's API and read how it achieved similar behavior for other platforms, which was that, it first of all found the `HOME`(environment variable) for the OS and then located corresponding path(s) for saving state in dir/file(s), turned out, that the crate was identifying `HOME` wrong only for android file-system. After this was diagnosed, we wrote a custom API that found `HOME` env var on all platforms irrespective of their OS (*[see this patch]*), by this API we were able to access the *app-specific private* directory and save state there. It was quite challenging, but we figured it out! Everything related to FFI layer, after this fix was quite easy. We eliminated the problems that existed in FFI layer via refactors and some modifications in functions and then, it built green! After fixing the FFI layer we wrote the CI for it that makes sure it builds each time a commit is pushed to any MR or branch and we can see the build status in pipelines too. Writing the CI for android-support crate was not a cakewalk, actually the application needs cross-compilation of our Rust library in order to function, so we need to make sure that the library which application is going to use on android devices is really compatible with android platform and by virtue we compile it on our PCs directly so that doesn't work quite, to make expected behavior happen there are two options

- Compiling the Rust library codebase on android-device(less feasible), *or*
- Cross-compiling the library on our PC/CI runner via providing support tools for android components

So quite obviously we went with the second option, for this we installed rust and android compatible components in our runners during the CI runtime and then compiled the library via checking out to the correct directory. But since, for compiling Rust library in each CI run we had to install the components and this specific pre-compilation part(or better to say setup portion) consumed a considerable portion of our CI script(and made it look a bit daunting too), so we packaged these components to our custom docker image and pulled it each time in our CI runs, this made our life easy and scripts beautiful :)

**NOTE:** If you don't have much idea of cross-compilation, then you can have a look at *[this awesome blog-post]* by Milan. It gives a clear understanding to the reader what cross-compilation is, irrespective of their previous knowledge on the same(yes, but basic knowledge on compilers is needed *a bit*). Spoiler alert: That *someone* in the opening of blog post is me :playful:

Also, since the refactor was incomplete in our android application codebase and the android-support crate so between these to big tasks I fitted this small refactoring, as a light break :P
You can see the MRs for them here:

- Fixing the FFI Layer: *[we/irdest/!31]*
- Refactoring the android-components: *[we/irdest/!32]*
- Adding the android-support crate to our CI Pipelines: *[we/irdest/!33]*

## III. UI Improvements

After fixing issues in android application and our Rust library, and writing a robust end-to-end CI for them, we moved forward towards improving the UI of the application. Previously the application used legacy design components and ideology, under this task we modernized these UI components and followed the material design guidelines(*[material.io]*), that improved the overall look of our Authentication screens. There is nothing much to explain in it as it was really quite easy to achieve and also we didn't encounter any problems.
You can see the related MR here: *[we/irdest/!26]*

## Acknowledgements

Well, by far it has been the most exciting summer for me and I had interesting experiences working on the project. Fixing components, was very difficult at the beginning due to many reasons one I would mention is the huge codebase which we have and it is not easy to learn about the functionality of each component present in it in short time, and also everything is intertwined too at many places. Going through it and fixing issues would definitely not have been possible without the immense support from my mentor, Spacekookie, who was always there to help me out and direct what to do. Their advices greatly helped in speeding up the development process and they are also a source of inspiration to me. Most importantly, Milan, who is not officially my mentor but has helped me a ton of times in technicalities of CI and setting up Nix environment(which I initially used for cross-compilation) about which I knew nothing, and many more instances. It won't have been easy for me to accomplish aforementioned tasks without direction and help from Kookie & Milan.
Thanks again to both of them :) and I'm more excited to work on the project with them further! :pray:

Cheers Until next time we meet!
∼Ayush Shrivastava /