How to run server code

The server file is located in a folder called server and the file is named server.py

The recommended python version is 3.10.6

Package version are specified in the requirements.txt file (if running the gunicorn server, you will also need to pip install gunicorn, I had version 20.1.0)

Run either of the following commands when inside the server folder, these will launch the server on port 8080, you can change the port as required. Ensure to port forward if you want to access the server from outside your local network. An example server is available at http://sdd.syedahmad.tech:8080, this server is already partially filled with data, a good example account to login to is username: og password: 000. As of 01/06/2023 this server is able to be accessed while connected to the DETNSW network. This server is known to sometimes crash under excessive load.

gunicorn --bind 0.0.0.0:8080 server:app

flask --app server.py run --host=0.0.0.0 -p 8080

After having changed the port you will need to update the main_networking.dart file on line 6 with the new ip/hostname and port. To see changes within the app, you will need to compile a new version.

Running insomnia API test suite (this is not essential and is only here for documentation purposes)

The file which can be imported into insomnia (Download - Insomnia) is in the server folder and titled "insomnia.json". The tests assume a completely fresh server has been started. To get all tests to pass you will need to change some values to a valid milliseconds since epoch time within the last week (with week starting at Sunday GMT). You can obtain such a number from the following link: Current Millis - Milliseconds since Unix Epoch. The parameters that need to be updated are:

Test: "Runfun sendrun TEST", Data changed: change the "start" parameter to the time obtained above.

Test: "Runfun getrun TEST", Data changed: change the name of the file requested such that the file name is the epoch value, with an extension of ".json".

Finally go to the TEST tab, then go to the "Get data from that run" test. Within the test code, change the expected start parameter to the epoch value.

If changing the start run time, inconsistent results may manifest if this run is viewed within the app UI.

If you have changed the server url that will also need to be updated throughout the tests.

How to run the app code (this assumes you are using a physical android device with Android 13 and are using a Windows computer)

If you want to simply see the code getting compiled without compiling it yourself, you can run the code at Pipelines - Runs for HackintoshwithUbuntu.runfun (azure.com) by clicking run pipeline. Check the build artifacts for the apk output.

1. Install the Flutter SDK (more detailed instructions can be found here Install | Flutter), Flutter SDKs can be found here Flutter SDK archive | Flutter – Note: I used sdk 3.11.0-0.1.pre linux, but most beta/stable versions past 3.10 should not present any issues)
2. Add Flutter to PATH
3. Install Android Studio
4. Install the Google USB driver
5. Enable developer options on the android phone and connect it either with a cable or wirelessly to a computer (connecting wirelessly will usually require typing adb commands and possibly connecting via cable once before doing any wireless connections)
6. Install VS Code (or utilise an existing installation)
7. Install the Flutter extension
8. Open the code folder in VS Code and wait for Flutter to fetch packages (if they do not automatically fetch you may need to run “flutter pub get” in the terminal)
9. Select the correct device from the selector in the bottom right
10. Press the play button in the top right (the command “flutter run” should also work, more detailed information is available at Test drive | Flutter)
11. If compiling in release mode you may need to create a launch.json (the command “flutter run --release” should also work)

Note: packages still use the old name of “runfun” for convenience

Android will sometimes automatically give the app blocked notifications permission and decide to refuse my requests to ask the user. There is nothing I can do about this (once Android decides your app cannot have notifications, you cannot even request them anymore) and if notifications are blocked and not requested on first start, they will have to be enabled manually. Due to the nature of Android workmanager, the notifications may come after a significant delay from the scheduled time. Unfortunately, this means that notifications are somewhat unreliable and they depend on how a device manufacturer has configured their background process optimisation, in my experience they perform somewhat better in debug mode.

When running the code, there may be a number of warnings/errors about deprecated APIs, this is on purpose, the new APIs have regressive functionality (see https://issuetracker.google.com/issues/180218747)

If you have errors

When running the code, any required SDKs or NDKs should be automatically downloaded but if they are not, please ensure the following are installed

- SDK for Android Tiramisu
- SDK for Android 31
- Android SDK build tools 34-rc4 (for versions 33.0.1 and 30.0.3)

- NDK (side by side) (25.2.9519653 and 25.1.8937393)
- Android SDK Platform-Tools 33.0.3
- Google USB driver

I have included a list of the versions of the libraries I used when compiling in “Known working package versions.txt”. Newer versions of Rive common have been especially known to cause issues when compiling.

Ensure your device appears when running the command “flutter devices”, run the command “flutter doctor -v” for more info about your devices and any possible issues

Run “flutter clean” then run “flutter pub get”

Deleting the build directory may also fix some issues