Showdeck Replay: Operation Manual

Hrólfur Gylfason<hrolfur20@ru.is>

Stefán Laxdal <stefanl20@ru.is>

Sverrir Þór Sigurðarson <sverrirts20@ru.is>

T-404-LOKA

RU Science and Engineering

Instructor: Haukur Hafsteinn Þórsson

Examiner: Ásgeir Freyr Kristinsson

11th May, 2023

1 Before getting started

This project uses Flutter, so start installing Flutter onto your machine before anything else. We'd recommend following Flutter's guide on that here.

Building and running on some platforms requires some extra setup besides installing Flutter. If you see the platform you want to build for listed below, check them before getting started for it guide before running in debug or building for production.

1.1 Web

If you are building for or running on the web, you must run the bash script `patch_web_build.sh`. This script modifies some auto-generated files to make them compatible with the web.

After doing this, be careful not to commit and undo these changes when you want to build for other platforms. If they are accidentally committed, you can either revert the commit through git or delete all the `*.g.dart` files modified in `patch_web_build.sh`.

1.2 iOS

If you want to debug on or build for iOS, we recommend following the Flutter iOS build and release guide.

2 Running in Debug

Compiling the Flutter app is relatively simple. Start by running this command:

```
flutter run
```

This command will then prompt you to select the platform you want it to run on if you have multiple platforms available. If you do not see an option for the platform you would like to test on, try running the following command, it should see why that platform is unavailable.

```
flutter doctor
```

3 Building for Production

Run `flutter build` to see the platforms you can build for. This command will show all the available subcommands, which allow you to pick what platform you want and what kind of build you want. For an Android APK, do `flutter build apk`.

3.1 Additional steps for Android

Because we are using app links on Android to skip the disambiguation dialog, you need to get the key file from Hrólfur or Stefán when building for Android, downgrade the app links to regular web links by commenting out the `signingConfig signingConfigs.release` line at `android/app/build.grade` or add your own key in `android/key.properties` and update the sha256 cert fingerprints in `web/.well-known/assetlinks.json` accordingly.

4 Build Runner

In Flutter, we use a few packages that utilize auto-generated Dart code. These packages are Isar, Riverpod, Freezed, and Ferry.

4.1 When should I run the builder?

If you change any class or method with `@something` above it, you are modifying something that utilizes auto-generation and should run the build runner to ensure everything is up-to-date. Modifying any GraphQL files also requires running the build runner, but it has more complexities that we will go into here below.

4.2 How do I run the builder?

The following command will run the build runner once:

```
flutter pub run build_runner build --delete-conflicting-outputs
```

However, suppose you are fixing something with more than one simple change. In that case, we recommend using the `watch` mode instead, as it runs the build runner automatically after every change and only runs it on the changed files reducing build wait times. The following command runs the build runner in `watch` mode:

```
flutter pub run build_runner watch --delete-conflicting-outputs
```

4.3 GraphQL

When modifying any GraphQL file, the build runner needs to be run as detailed above. However, if the changes rely on changes to the GraphQL schema from the server, you also need to update the schema before running the build runner like this:

```
npx get-graphql-schema https://api.showdeck.io/graphql > lib/src/graphql/schema.graphql
```

When running this, make sure you are at the root of the project, as it relies on relative paths.

If you get a `[SEVERE] ferry_generator:...` and somewhere around that a `Bad state: No element`, then this means that Ferry failed to generate the SQL types and bindings for the queries. This is usually caused by either using the wrong name in some GraphQL files you were modifying or not having the most up-to-date schema you expected. In such a case, you should run the command shown above.

5 Code Style

5.1 Frontend

We use Effective Dart, Dart's official code style. This style applies to all sorts of naming, such as variables, classes, files, and lots of other things, like sorting imports.

To help us follow this code style, we also use Dart's official linter. You can get all lints in the project by running `flutter analyze`, but this should preferably be put into the editor to give these warnings as you are coding. We also have a custom rule for warning about unawaited futures, which can be a common source of bugs. Additional custom lints can be added to the file `analysis_options.yaml`.

Last, we use Dart's formatter, which Effective Dart requires. This formatter ensures that all our code looks similar and makes it easier to work with each other's code.

5.2 Backend

In the backend, we follow Python's PEP8, which is Python's official code style, and it sets rules around formatting in Python. To help us follow this, we use the code formatter Black to format all our Python code.