ITO Flutter developer guidelines

As a potential future part of the ITO Flutter development team you as the developer should be at least familiar with the concepts mentioned in the rest of this document. Of course the level of your understanding may vary, but you should use this document to guide your research in the domain of mobile and web development with Flutter.

Table of contents

OOP and Dart

As a modern developer you should be familiar with some of the basic concepts in OOP (Object oriented programming). Abstraction, inheritance, polymorphism and encapsulation are not just theoretical concepts but they are often used in flutter developers daily programing flow, so everyone should expand their knowledge on these concepts to better grasp some of more advanced concepts that are very important in our applications.

So the very first thing that you could do to become a more advanced flutter developer is to try to research and understand the basic concepts of OOP, preferably using Dart as the language to try to implement some of the concepts that you just learned. Fundamental knowledge of the Dart programming language is required to be able to create flutter application.

At the very least you should understand the basic syntax of Dart, some of the most frequently used methods for lists, string, know what are maps in Dart and what is the difference between Future and Stream and where are they used.

State management techniques

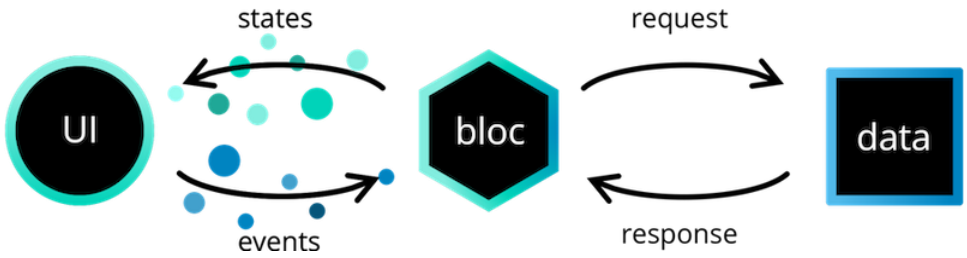
Every flutter application by default focuses on natural look and feel that is very easily implemented using existing (Material and Cupertino) or custom made widgets. Flutter by it's nature offers fully customizable UI that you can directly "draw" on the screen.

More often then not this UI requires a bit more complex logic behind so the user's experience is fluid and practical. Behind the scenes almost every flutter application manages, stores, sends and manipulates some kind of data set. This complexity of the state management requires some kind of standardization. In the Flutter community there are quite a few of state management techniques used today. No one can with 100% certainty say one technique is better then the other one, but you should carefully weigh the needs of your project and the complexity of that state management technique.

We, the ITO Flutter team, found the BloC pattern to be the most useful for many of our use cases and that is primarily our way to go.

Underlying logic is fairly simple, there are three main components:

- Event – represents some user or programmatic action(for example user click submit button)
- Bloc – represents the main engine for implementing the business logic based on input events
- State – represents the state of a logical part of the application at any given moment in time



High level overview of BloC pattern

Find out more about BloC pattern in the references in Useful documentation part of the document.

Commonly used packages and services

More often then not there is a requirement on the new project for a feature to be implemented that was in a way implemented on a previous project or a variation of that feature. So for this reason it is a good practice to write your code in a way that it is more generic, configurable and usable on multiple projects, keeping up with a good old DRY rule of programming (DRY stands for Don't repeat yourself).

Using the logic mentioned above we tend to maximally reuse our code either by using existing community approved packages or implementing and maintaining our own.

Some of most commonly used packages in out flutter projects:

- rest_api_client (https://pub.dev/packages/rest_api_client)
- storage_repository (https://pub.dev/packages/storage_repository)
- intl (https://pub.dev/packages/intl)
- catcher (https://pub.dev/packages/catcher)
- location (https://pub.dev/packages/location)
- provider (https://pub.dev/packages/provider)
- url_launcher (https://pub.dev/packages/url_launcher)
- octo_image (https://pub.dev/packages/octo_image)
- fluttertoast (https://pub.dev/packages/fluttertoast)
- flutter_bloc (https://pub.dev/packages/flutter_bloc)
- flutter_html (https://pub.dev/packages/flutter_html)
- flutter_svg (https://pub.dev/packages/flutter_svg)
- image_cropper (https://pub.dev/packages/image_cropper)
- photo_manager (https://pub.dev/packages/photo_manager)
- image_picker (https://pub.dev/packages/image_picker)
- carousel_slider (https://pub.dev/packages/carousel_slider)
- pull_to_refresh (https://pub.dev/packages/pull_to_refresh)
- webview_flutter (https://pub.dev/packages/webview_flutter)
- flutter_typeahead (https://pub.dev/packages/flutter_typeahead)
- connectivity_plus (https://pub.dev/packages/connectivity_plus)
- permission_handler (https://pub.dev/packages/permission_handler)
- google_maps_flutter (https://pub.dev/packages/google_maps_flutter)
- cached_network_image (https://pub.dev/packages/cached_network_image)
- google_maps_webservice (https://pub.dev/packages/google_maps_webservice)

ITO Flutter Project sample

ITO's Flutter development team believes above all in the power of standardization and good practices. So the first logical step was to create an example project with the production ready set up architecture and commonly used features implemented. Our team is continuously working on improving this project and we welcome and rely on the community's contribution to this open source project.

You can find the source code of this project on our Github repository: https://github.com/itodjel/flutter_edu/tree/master/flutter_boilerplate

The project is supposed to showcase some of the most commonly used architectural patterns in flutter application, the state management is implemented with flutter bloc library and the app is supposed to conform to the latest standards in flutter community.

In the same repository, right beside the flutter_boilerplate project you can find few of our flutter challenges that are designed to be implemented by anyone who wants to test or showcase their knowledge. You clone the repository, choose the challenge, solve it and make a pull request back to our main repository. Our team would be thrilled to review it and be fascinated by your skill 😊 or maybe suggest some improvements, so feel free to collaborate at any time, either by submitting solutions for challenges or improving our template project(flutter_boilerplate).

Advanced topics

From the very beginning Flutter was created with the simplicity and reusability in mind, using one codebase to create apps for multiple platforms is the core of this framework.

An advanced Flutter developer should at least understand or be familiar with some of the concepts bellow:

- What is dependency injection and how it is implemented in flutter project, are there alternatives for dependency injection like for example service locator pattern, how it is implemented and used in flutter projects.
- What is the use of streams and where are they most useful
- What are flavors in android and what schemes in iOS when building an application
- The art of publishing iOS application 😊
- Communication between flutter app components

Useful documentation

- Flutter BloC library: https://bloclibrary.dev/#/gettingstarted?id=overview
- Flutter team playlist: https://www.youtube.com/playlist?list=PLjxrf2q8roU2HdJQDjJzOeO6J3FoFLWr2
- Widget of the week: https://www.youtube.com/playlist?list=PLjxrf2q8roU23XGwz3Km7sQZFTdB996iG
- Flutter tutorials: https://www.youtube.com/c/ResoCoder/videos
- Flutter zero to hero: https://www.youtube.com/c/Flutterly/featured
- Flutter docs: https://flutter.dev/docs