Huzefa Mandviwala
Charles Thao

EC500 Case Study Project - Flutter

Flutter is an SDK developed by Google for creating mobile interfaces for applications easily and quickly. Flutter is an open-source project used in many applications, and its goal is to enable developers to craft interfaces that function well, look good, and are not complicated. Some of its core features include a fully customizable and flexible UI, a responsive interface, and integration with existing tools like Java, Kotlin, Objective C, and Swift code, platform APIs, and 3rd party SDKs.

Technology and Platform

Flutter is written in a language called Dart, a programming language also developed by Google. According to the project site, Dart was "designed to be easy to write development tools for, well-suited to modern app development, and capable of high-performance implementations." The main purpose of Dart is to build web, server, and mobile applications.

If we were to write Flutter now, we think that Dart is still a good choice for a few reasons. First of all, Flutter is a Google project using primarily Google libraries and frameworks (such as Skia), so it makes sense to use a Google-developed language such as Dart. While there exist other more established languages for creating applications, such as Javascript, Dart has a few more modern features that give it the upper hand. Dart includes a rich type system, module system, native testing framework, and a native package manager.

- Build system
    - Build tools/environment:
    - To run in mainstream web browsers, Dart relies on a source-to-source compiler to JavaScript. When running Dart code in a web browser the code is precompiled into JavaScript using the dart2js compiler. Compiled as JavaScript, Dart code is compatible with all major browsers with no need for browsers to adopt Dart. Through optimizing the compiled JavaScript output to avoid expensive checks and operations, code written in Dart can, in some cases, run faster than equivalent code hand-written using JavaScript idioms.
    -

Flutter also supports using C/C++ libraries, including direct calls from C/C++ to Dart and from Dart to C/C++. This is one feature that Flutter is looking to improve upon in the coming years.

For graphical applications, Flutter makes use of the Skia 2D Graphics Library. The Skia Library enables features such as complex text rendering, shaders, and path effects. Skia was also developed by Google.

Testing

Flutter enforces a strict testing regiment in order to maintain sanity and working code. Dart tests are written with a “flutter_test” package API, follow a specific .dart naming convention, and placed inside a master/dev/tests/ subdirectory. Flutter supports regular unit tests, unit tests with golden-file testing, and end-to-end tests. Regular unit tests written with the flutter_test package utilize flutter-specific extensions on top of the Dart test package in order to test against specific Flutter functionality. Golden-file tests involve comparing an interface generated by Flutter code to a screenshot taken manually of the expected output. This comparison is done at a pixel-by-pixel level in order to ensure that the output generated from the code is an exact match to the expected output. The screenshot is known as the “golden-file”, because it is the standard to which the generated output must match up to.

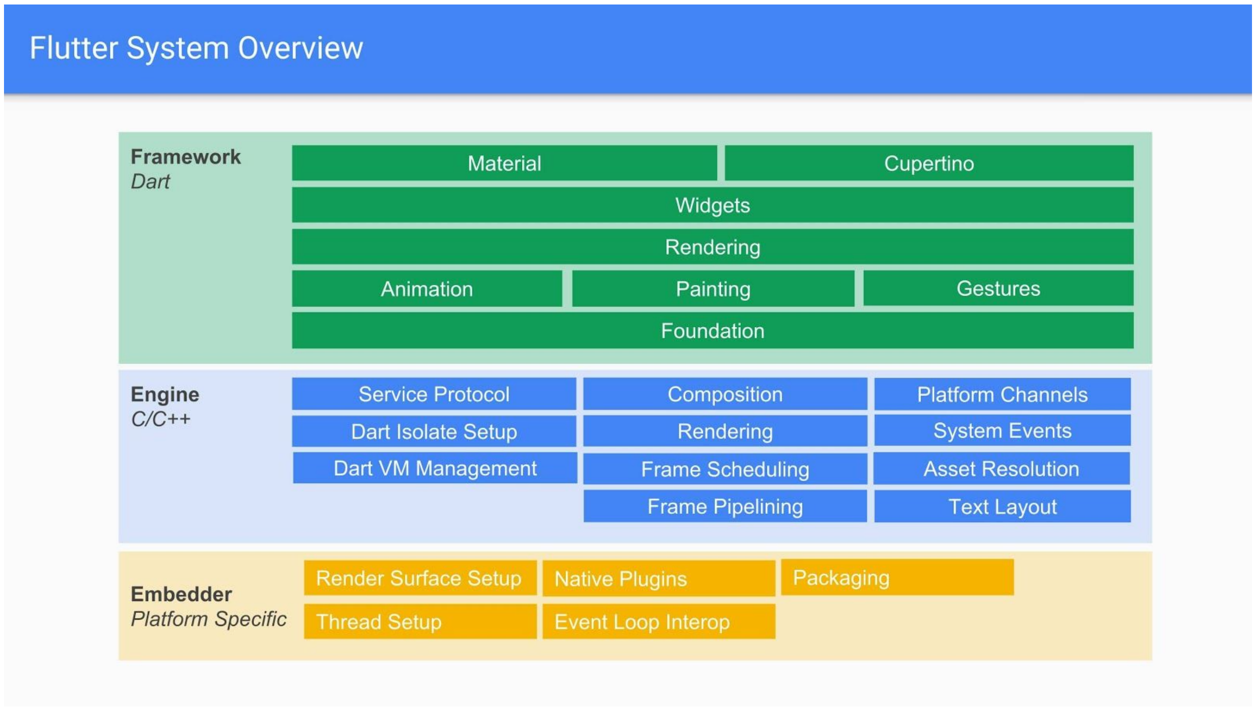
While these tests are mandated by Flutter, developers also include integration tests, manual tests, and missing dependency tests. Each of these can be found in their own directory under master/dev/. All these tests ensure that new features and bugfixes will work with existing code and not misaffect or break any existing code.

These tests, all included in the test.dart script for each PR, are run by Flutter’s Cirrus Continuous Integration platform. Cirrus is a modern CI, built by Google, that is cloud-enabled in order to perform fast, efficient, and secure testing. Cirrus employs bots to run testing on Flutter tools, framework, rebuilds, and updates. It also uses containers to test the various tasks and builds. The Cirrus configuration can be found in master/.cirrus.yml, and we gathered from this file that Cirrus is testing on Windows, Mac, Linux, Android, JDK, and iOS.

Finally, there are post-commit end-to-end tests that run on a physical lab known as the Devicelab. The Devicelab tests Flutter on physical Android and iOS devices using some scripting tasks, and reports the results back to developers. This final testing stage ensures that Flutter actually works on real devices as a whole package. The Devicelab employs 2 Linux agents, 7 Mac agents, and 2 Windows agents. Each agent runs various tasks in order to give detailed analysis of what exact task might fail on what exact system.

Flutter ensures test coverage using Coveralls, although this feature is currently broken. Instead, they employ a few workarounds to download coverage information. In addition, developers are encouraged to check code coverage of any new features or bugfixes implemented before submitting any PRs.

Software Architecture



Contributing to Flutter is a regulated process with a few crucial steps. After setting up an engine development environment which uses C++, Java, and Objective C, a developer must also setup a framework development environment which uses Dart. Flutter enforces strict Tree hygiene and Issue hygiene, which describe how to handle changes, failures, triaging/assigning bugs, and Github labeling/naming conventions. Finally, Flutter also has a comprehensive style guide which dictates code format, syntax, and API design.

In general, a workflow for adding a new feature might look like the following:

1. Fork the repo on Github
2. File a new issue (if one does not exist) for the new feature
3. Discuss design on the issue with the Google Flutter team
4. Create a branch on your fork, implement change
    a. Ensure the code is tested
    b. Use code coverage tools to check that new code is covered by tests
    c. Follow style guide for formatting
5. Submit branch as a PR
6. Code review by Flutter experts
7. Ensure PR passes pre-commit tests

    a. Also test post-commit tests locally (Devicelab tests)
    b. Wait for Cirrus to successfully complete CI
8. Merge your new code!
9. Watch post-commit Devicelab tests to ensure everything works
    a. If anything breaks, revert and study problem

To use Flutter to develop your own application, you can import the package in a Dart file. Flutter provides a base project structure to get started with, including a main.dart file, tests directory, and other boilerplate files. The sample application that Flutter provides is a stateless, simple application with a button and some text. You can add widgets, text, and functionality to the sample application to start developing in Dart. Flutter cannot run on its own, because it is a development package. One unique feature of Flutter is the “hot reload”, which allows you to load an updated version of an app or widget onto the simulator (e.g. iPhone X emulator) very quickly, rather than restarting the entire emulator just to test a small change in the code.

- Asynchronous parts of software
- Diagrams
- Separation of concerns and information hiding
- Architectural patterns
- Object oriented vs functional components

Two Defects

- Description of each
- Architecture change vs adding new functionality vs ?
- Patch/PR to fix problem/add functionality
- Contributor guide: https://github.com/flutter/flutter/blob/master/CONTRIBUTING.md

1. VM crash

Issue link: https://github.com/flutter/flutter/issues/29379
Description: VM crashes while loading and resizing images in Flutter Android app (Multiple mutators entering an isolate / Dart VM is shutting down)

Potential
2. iOS compatibility
Issue link: https://github.com/flutter/flutter/issues/29437
Description: I'm stuck at debugging flutter sample apps on physical iOS device (iphone 6s - iOS 12.1.4) using android studio. I managed to debug using Xcode directly. Also, debugging apps on simulator seems to be working fine using android studio.