Flutter Report

Nicolás Hernández 201412420
Rogelio García 201326488

THE FINAL REPORT WILL BE AN APK WITH THIS INFORMATION

Intro

We are Rogelio García and Nicolás Hernández.

Welcome to our report!

To browse this padlet-like report, swipe down to read each topic and swipe right to see another one. Tap on the button on the bottom right of the screen to see further details.

If a card has three dots underneath, like this one, tap on them to see references or further information!

Repository::https://github.com/Braitt/flutter_report

FOTO NICO

Apps chosen

As flutter is a pretty new thing, finding an app that has over a million downloads on the Play Store is not easy, in fact, only only found Alibaba with 50 million downloads. The second biggest app would be Reflectly with 100.000. Moreover, finding an app that is open-source is even harder.

Given that we couldn’t find an app that met these criteria, we chose two open-source apps and will be doing different analysis on them: 2T2000s and SpaceX GO!.

Being limited in both a big open-source development ecosystem like, for example, Firefox’s, and number of features, these apps are still good examples for certain things that will be covered in the analysis; despite being small, they implement design patterns, animations and other interesting things that will be covered in this report. They also appear featured in Flutter's official website and on Medium.com.

Flutter showcase

Medium showcase

2T2000s

This app was created for Trinity College at University of Toronto as an app to guide students through the orientation week. It serves as a schedule where students could see when and where activities are due. People can also upload pictures and view others' pictures. This app showcases breathtaking animations in every Widget and background.

Info

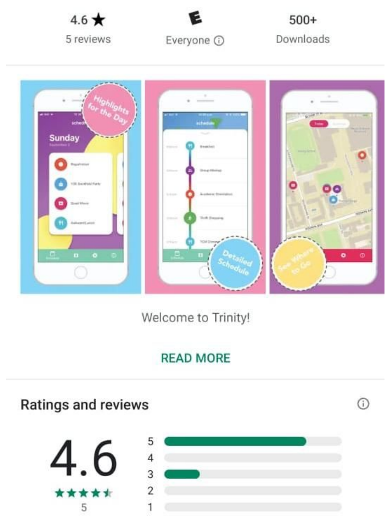
Finding technical details about this app is not possible. On Github, there are 7 commits.

6 commits were done the 30th of September 2018, including an initial default commit (when the repository was created), four commits each updating the README.md file with new images and one commit containing the whole app.

The other commit was done the 11th of October; it updates the license.

Despite having only four tabs and under 20 views, this app has over 7500 lines of code and 113 classes, most of them Widgets, or related to custom-made animations

The project was done over 3 months by a single person, Matthew Tory, and has since then been displayed on Flutter's webpage.



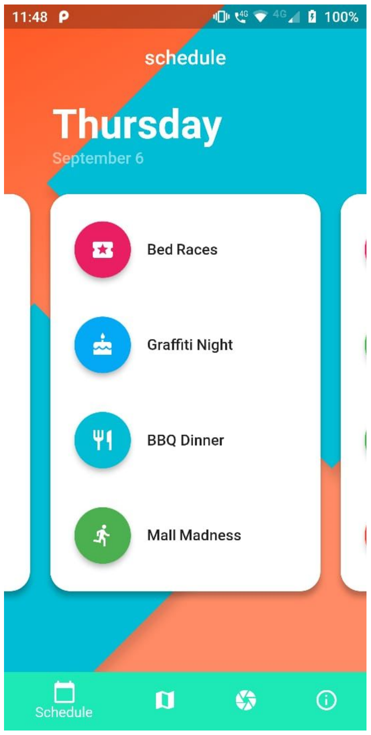
This app has only 5 reviews and has been downloaded over 500 times. The average rating is 4.5, with only 1 rating being 3 stars.

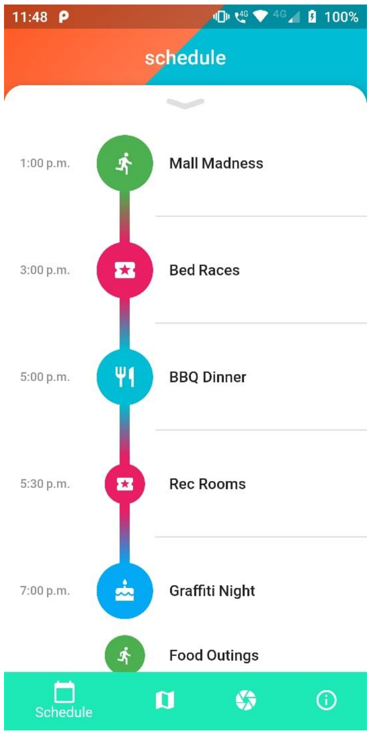
Features

2T2000S app provides the following features:

- Review daily scheduling and activities.
- Review day activities detail on timeline.
- Review activity information (location, time, map, description).
- Look for activities using a map.
- Retrieve photos from other students.
- Add new photos with description and overlays.
- Save locally photos and delete them.
- Review more information about app like resources, questions and contacts.

Some screenshots



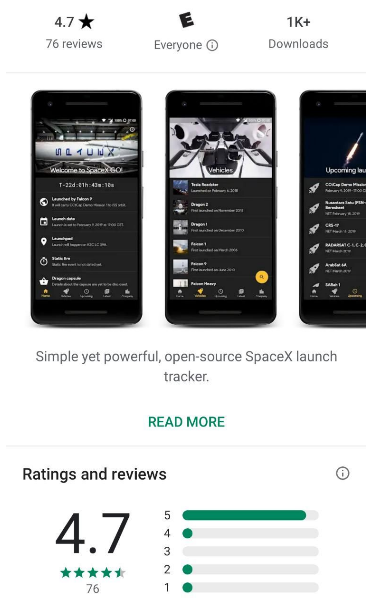


SpaceX GO!

This app works as a tracking for SpaceX past and upcoming launch dates and news, as well as information about the vessels and the company. It's an open source app that was actually built for fun and for educational purposes, but has since then become a referent for Flutter development, and has gained modest traction in terms of activity on Github.

Info

On github, SpaceX GO! Has over 400 commits, several pending issues, with a new issue being posted every couple weeks. It has about 3000 lines of code written and 80 classes, about half of them Widgets.


4.7 ★ 76 reviews | Everyone | 1K+ Downloads

Simple yet powerful, open-source SpaceX launch tracker.

READ MORE

Ratings and reviews

4.7 ★★★★★ 76

This app has received only 76 reviews total and over 1000 downloads and an average rating of 4.7. No ratings under 5 stars have a written review, so it’s difficult to determine what caused these users to rate it with four, two or one star.

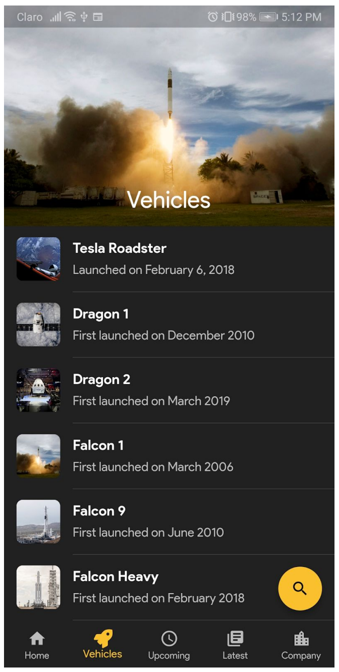
This app is available in both the app store and the play store, but we will only focus on the Android version of the app.

Features

SpaceX-Go app provides the following features:

- Review last launches.
- Search for a launch by name.
- List vehicles used by launches.
- Know more about company (SpaceX).
- Add launches to calendar.
- Review details of launches (Rocket, Payload and Vehicle).

Some screenshots



Tesla Roadster
Launched on February 6, 2018

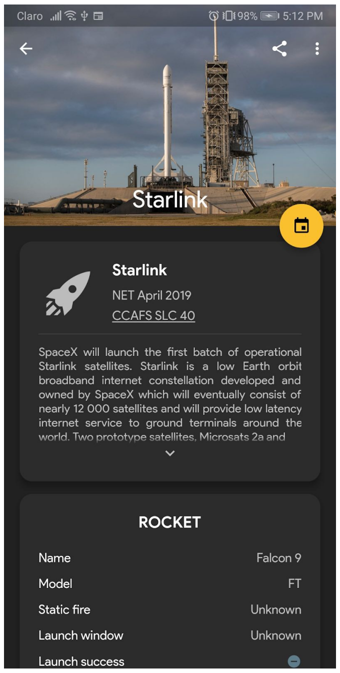
Dragon 1
First launched on December 2010

Dragon 2
First launched on March 2019

Falcon 1
First launched on March 2006

Falcon 9
First launched on June 2010

Falcon Heavy
First launched on February 2018



Starlink
NET April 2019
CCAFS SLC 40

SpaceX will launch the first batch of operational Starlink satellites. Starlink is a low Earth orbit broadband internet constellation developed and owned by SpaceX which will eventually consist of nearly 12 000 satellites and will provide low latency internet service to ground terminals around the world. Two prototype satellites, Microsats 2a and

ROCKET

Name	Falcon 9
Model	FT
Static fire	Unknown
Launch window	Unknown
Launch success	⊖

Analysis chosen

We wrote different reports based on most of the topics covered on the course. The complete list is:

- Performance, including profiling, multithreading and battery, memory and CPU management
- Storage and data handling
- Rendering, UI and animations, including accessibility
- Eventual Connectivity
- Code analysis: libraries and patterns used, native code implemented and platform specific code, good and bad programming practices. This report resembles a code review.

Doing dynamic performance profiling on 2T200s wasn't possible since the app can't be run. With the code at hand, however, we could do static profiling and more in-depth code analysis.

What we learnt

StreamBuilders

We found out way simpler ways of managing data when using a backend, namely Firebase and subscriptions to data documents and collections. In our project we implemented MVVM using

Scoped Models and RxDart, which were accessed by the widgets to fetch, add or remove data. Ultimately, however, we are mainly displaying data.

To display data, 2T2000s uses StreamBuilders, which ‘attach’ to a subscription and show data automatically. This is better done with the BLoC pattern, described in detail in the “code” report.

Programming experience

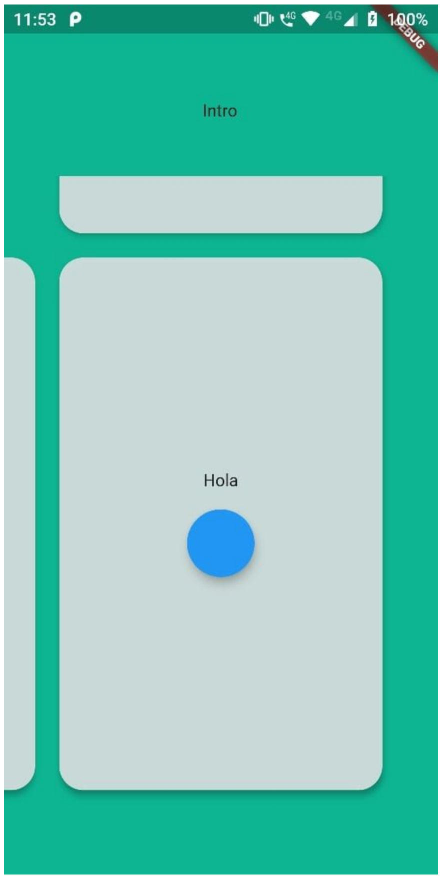
Static profiling using Dart Analyzer, built into Visual Studio Code is extremely good. Flutter’s error messages aren’t very informative for the developer when compared to Java’s for example, but the help that the Flutter and Dart plugins provide when developing make up for this.

On the other hand, the ecosystem in pages like StackOverflow isn’t nearly as huge as for other languages like Python, but the community is very active and in only a couple months since Flutter’s first stable release, it has already garnered a healthy amount of online resources, most of them in Medium.com.

Another lesson was the use of factories. In our project we used a great deal of factory methods (though we never implemented one), and for this report we actually had a case where the default constructor wasn’t enough and that’s where factories come in.

What we used in this app

For this app we started by making use of PageView widgets, like the ones used in 2T2000s, and we ended up having several PageViews in one PageView to achieve a map-like style.



assets/images/home/version1.jpg

We then decided this was too rigid since it didn't wrap around content, so we used Slivers in a similar fashion as 2T2000s does to achieve the final padlet-like style. We then made use of other widgets we already knew to build the rest of the app, including the background gradient.

Finally we added this little animation and gradient we're quite proud of. It also avoids the cutting shown on the screenshot earlier.

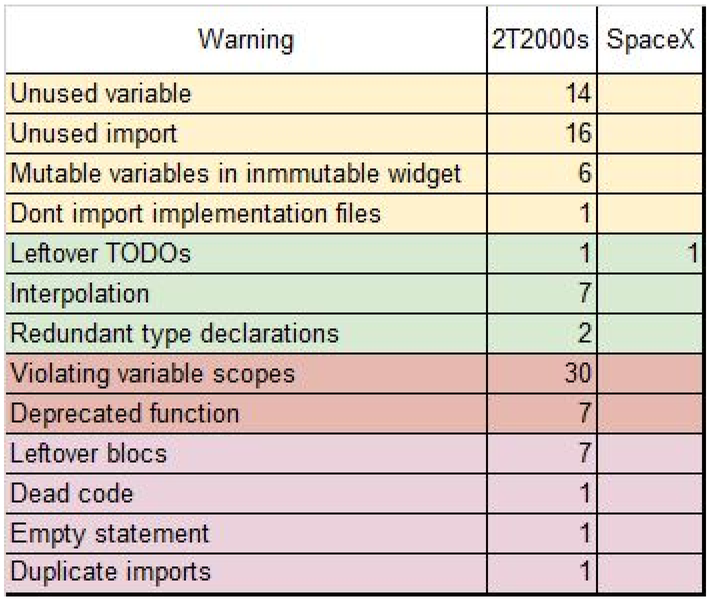
assets/images/home/animation1.gif

You can find the source code of this app in our repository!

Repository::https://github.com/Braitt/flutter_report

Performance

The static profiling tool used was Dart Analyzer. It is available on Visual Code Studio, included in either Flutter's or Dart's plugin. Having used this profiler for the project, we found that it's very useful in alerting errors and warnings opportunely.

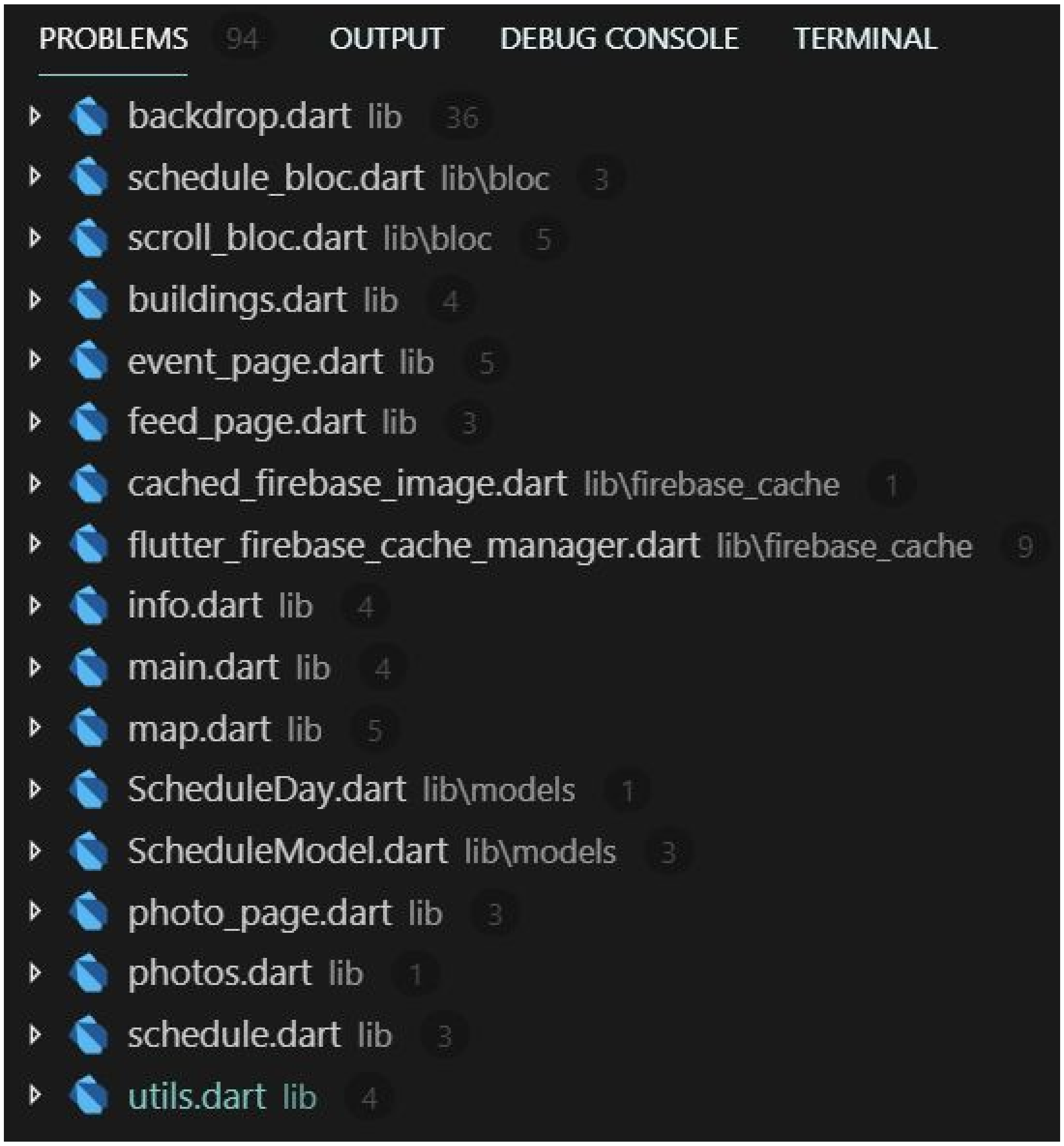


Warning	2T2000s	SpaceX
Unused variable	14	
Unused import	16	
Mutable variables in inmmutable widget	6	
Dont import implementation files	1	
Leftover TODOs	1	1
Interpolation	7	
Redundant type declarations	2	
Violating variable scopes	30	
Deprecated function	7	
Leftover blocs	7	
Dead code	1	
Empty statement	1	
Duplicate imports	1	

Static Profiling

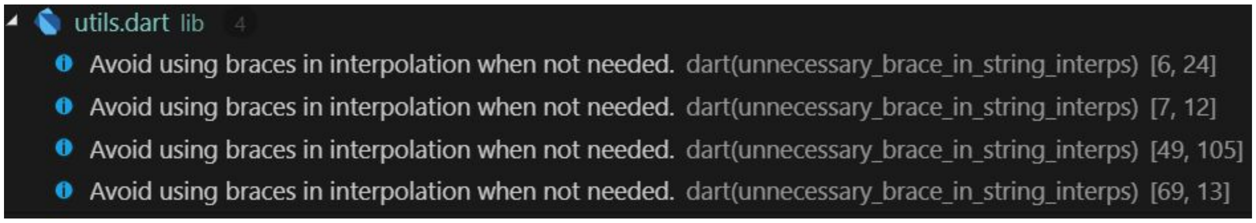
2T2000s

Static profiling detected 94 problems spread across 17 files.

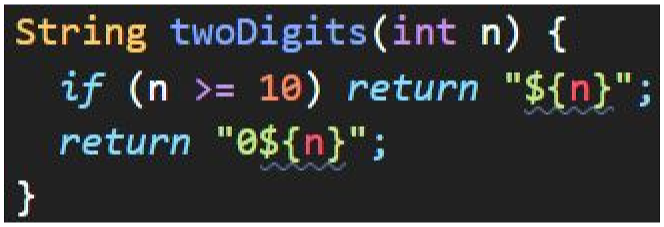


```
PROBLEMS 94    OUTPUT    DEBUG CONSOLE    TERMINAL

backdrop.dart lib 36
schedule_bloc.dart lib\bloc 3
scroll_bloc.dart lib\bloc 5
buildings.dart lib 4
event_page.dart lib 5
feed_page.dart lib 3
cached_firebase_image.dart lib\firebase_cache 1
flutter_firebase_cache_manager.dart lib\firebase_cache 9
info.dart lib 4
main.dart lib 4
map.dart lib 5
ScheduleDay.dart lib\models 1
ScheduleModel.dart lib\models 3
photo_page.dart lib 3
photos.dart lib 1
schedule.dart lib 3
utils.dart lib 4
```

Several of these are more about style: “Avoid using braces in interpolation when not needed.”, referring to using `"0${n}" instead of just "0$n"`



```
utils.dart lib 4
  Avoid using braces in interpolation when not needed. dart(unnecessary_brace_in_string_interps) [6, 24]
  Avoid using braces in interpolation when not needed. dart(unnecessary_brace_in_string_interps) [7, 12]
  Avoid using braces in interpolation when not needed. dart(unnecessary_brace_in_string_interps) [49, 105]
  Avoid using braces in interpolation when not needed. dart(unnecessary_brace_in_string_interps) [69, 13]
```


```
String twoDigits(int n) {
  if (n >= 10) return "${n}";
  return "0${n}";
}
```

Other ‘cosmetic’ warnings are about leftover TODO marks and not abiding the effective Dart guide.

For example, explicitly declaring variables to null instead of letting Dart assume its null value: \n\n\t- ‘var a = null;’\n\ninstead of just\n\n\t- ’var a;’

Also, declaring the type of variables in a class constructor when Dart can assume its type:\n\n\t- Class Point {\n\t\tInt x, y;\n\t\t Point(this.x, this.y);\n\t}\n\ninstead of\n\n\t- Class Point {\n\t\tInt x, y;\n\tPoint(int this.x, int this.y);\n\t}

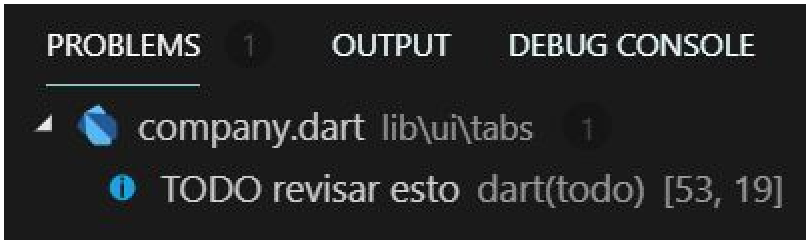
Other errors refer to good practices, namely “This class (or a class which this class inherits from) is marked as '@immutable', but one or more of its instance fields are not final: DayTitle.date”. This refers to having mutable variables (read: just regular variables without “final” or “const” keywords) in a StatelessWidget, which is inherently inmutable. In these cases, it’s better to mark the variables as inmutable with the “final” keyword. This doesn’t have real impact on the app execution.

Another error refers to accessing a protected variable from another class, violating the @protected tag. This is due to using outdated libraries; a function that was once public is now protected and deprecated in favor of another. Similarly, two classes that were implemented.

Other warnings do impact performance, or at the very least allocated memory. For example, unused variables, unused imports, unused libraries and unused classes. Also relating to imports, we found double imports in one file.

SpaceX GO!

Static profiling found no issues related to code. Only one issue was found, a left over TODO. It’s, funnily, written in Spanish and was left by a Spanish contributor.



```
company.dart lib\ui\tabs 1
  TODO revisar esto dart(todo) [53, 19]
```

Multithreading

2T2000s

This app has no actual multithreading, i.e., no isolates were used, and thus all the developed code runs in a single thread.

On the other hand, asynchronous calls are handled with simple placeholders, like using CircularProgressIndicators when there is no data available.

```
    StreamBuilder(
stream: Firestore.instance.collection('events').snapshots(),
builder: (context, snapshot) {
  if (!snapshot.hasData) return Center(child: CircularProgressIndicator());
```

SpaceX

This app uses no multithreading.

Memory & CPU

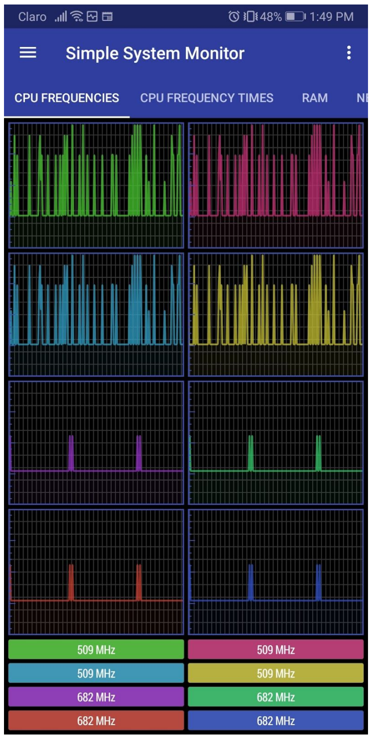
2T2000s

We used an application made for profiling general performance on a mobile device (https://play.google.com/store/apps/details?id=com.dp.sysmonitor.app&hl=en_US) and it helped us to measure CPU and memory usage. The following tests were made on a Huawei Mate 10 Pro (6GB RAM, 8-core (4-2.32 Mhz, 4-1.86 Mhz) and 128 GB Storage), and every app was open on a sandbox environment with no other processes in background but main operating system.

Tests took about 500 seconds as profiling app allowed. This information shows average frequency used by each one of cores and average bytes used of memory during the test. Main features of the applications were used and we tried not to open external applications (e.g. Youtube, web browser, etc.).

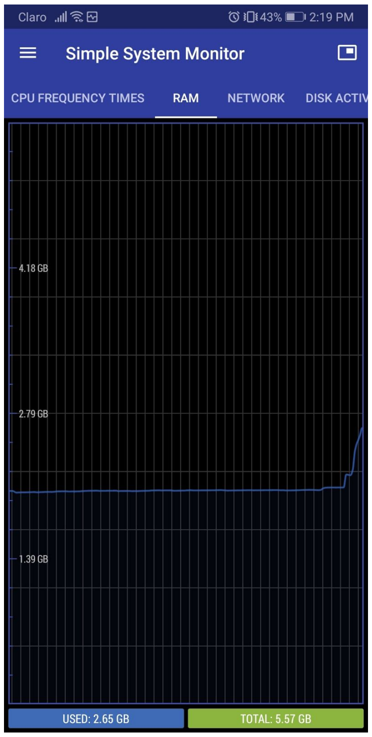
Initial state of device:

- CPU usage:



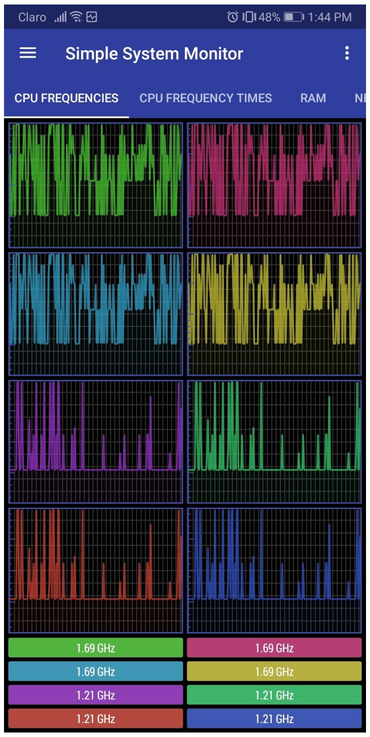
509 MHz | 509 MHz
509 MHz | 509 MHz
682 MHz | 682 MHz
682 MHz | 682 MHz

- Memory usage:

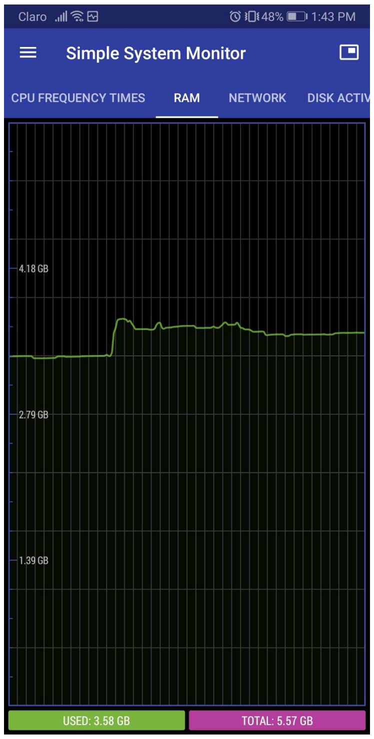


2T2000S state after use:

- CPU usage:



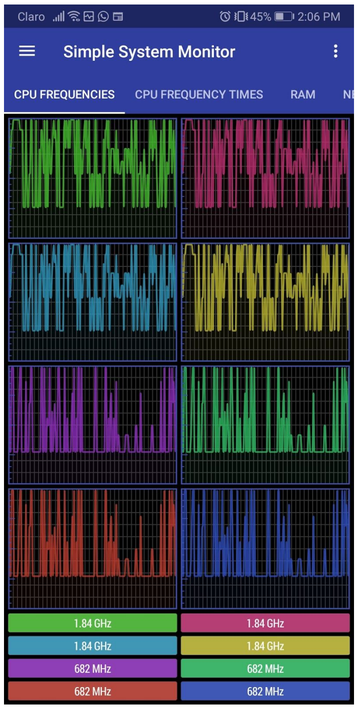
- Memory usage:



SpaceX

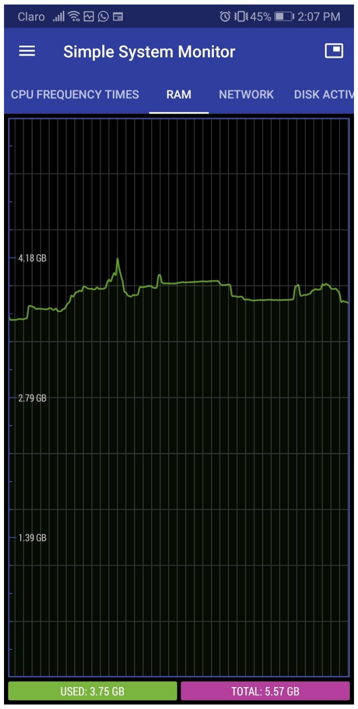
SpaceX-Go state after use:

- CPU usage:



1.84 GHz
1.84 GHz
1.84 GHz
1.84 GHz
682 MHz
682 MHz
682 MHz
682 MHz

- Memory usage:



USED: 3.75 GB
TOTAL: 5.57 GB

Dynamic profiling

SpaceX Only!

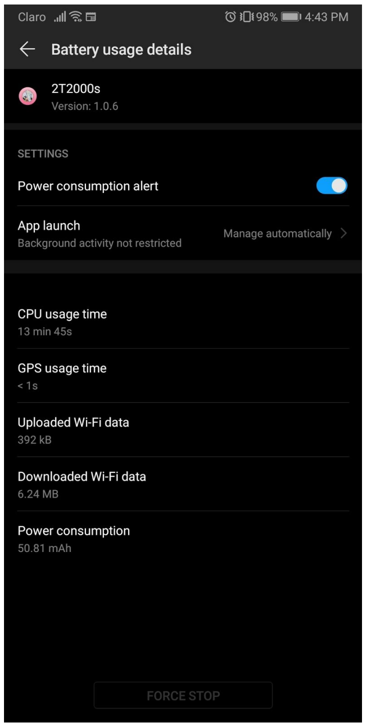
This app was run on debug mode to find out insights about it’s performance.

Battery

The following tests were made on a Huawei Mate 10 Pro (6GB RAM, 8-core (4-2.32 Mhz, 4-1.86 Mhz) and 128 GB Storage), and every app was open on a sandbox environment with no other processes in background but main operating system.

The test was to use the application for 15 minutes and observing the energy expenditure in the device. The battery usage information was provided by operating system native features that measured spended energy by app.

2T2000S state after use:



2T2000s
Version: 1.0.6

SETTINGS

Power consumption alert: on

App launch
Background activity not restricted
Manage automatically

CPU usage time
13 min 45s

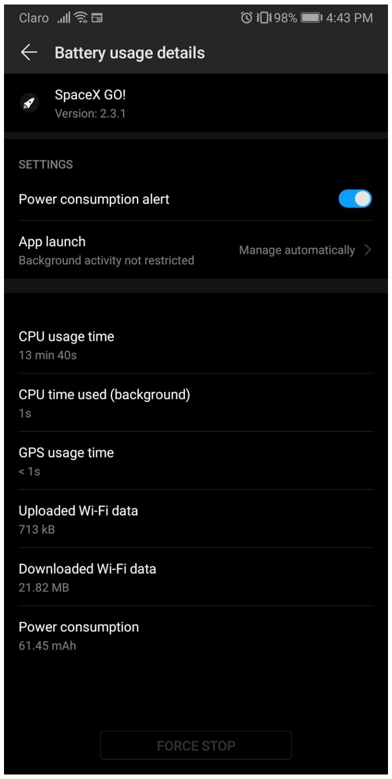
GPS usage time
< 1s

Uploaded Wi-Fi data
392 kB

Downloaded Wi-Fi data
6.24 MB

Power consumption
50.81 mAh

SpaceX-Go state after use:



Claro 98% 4:43 PM

Battery usage details

SpaceX GO!
Version: 2.3.1

SETTINGS

Power consumption alert

App launch
Background activity not restricted — Manage automatically

CPU usage time
13 min 40s

CPU time used (background)
1s

GPS usage time
< 1s

Uploaded Wi-Fi data
713 kB

Downloaded Wi-Fi data
21.82 MB

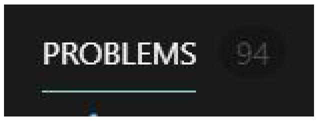
Power consumption
61.45 mAh

FORCE STOP

On one hand, 2T2000S did not use CPU on background unlike other apps (Facebook, Instagram, SpaceX-Go, ...). Also, the power consumption is 90 mAh (average) every 15 minutes in this device using common apps, and this application used 50.81 mAh (55% less). On the other hand, SpaceX-Go app did use CPU on background and consumed 11 mAh (20% higher) more than 2T2000S. This may be because SpaceX-GO makes a greater number of queries to obtain the information and render them.

Conclusions

Several of the errors found are fixed simply by using Dart's Static Analyzer. In Visual Code Studio, for instance, two commands can be used to automatically fix import errors and fix simple code styling, and warnings can be seen on a tab with links to the line of error.



Storage & Data Handling

This section describes how data is stored in the device and presented to the user.

1. Use of SharedPreferences
2. Use of local storage
3. Use of remote storage
4. Cache & Data Fetching Strategies

This also covers a bit about how the app handles eventual connectivity, but that is further discussed in the “Connectivity” report. It also covers a bit about how the app actually saves images (or attempts to save them) but that is discussed in detail in the “Code” report.

SharedPreferences

The only use of Shared Preferences is done by the cache manager. This is how the implementation of the Flutter Firebase Cache plugin works, but it will be included here since the library was implemented anew.

Everytime a cached file, in this case only photos and filters, is requested, the cache is updated in the CacheManager instance and also in Shared Preferences, in the form of a Json string.

Separately, the timestamp of the last update to the cache is also saved in Shared Preferences as an int (milliseconds since epoch).

Local — Private

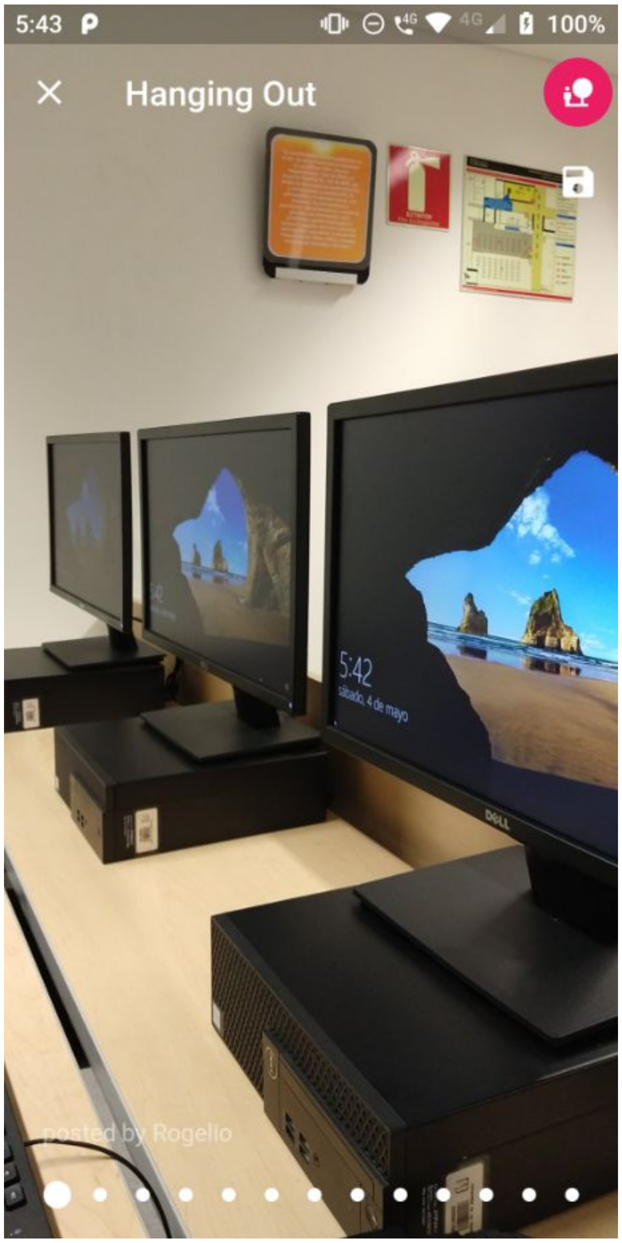
This refers to storing data in SQLite. This app does not use this storage strategy.

External — Public

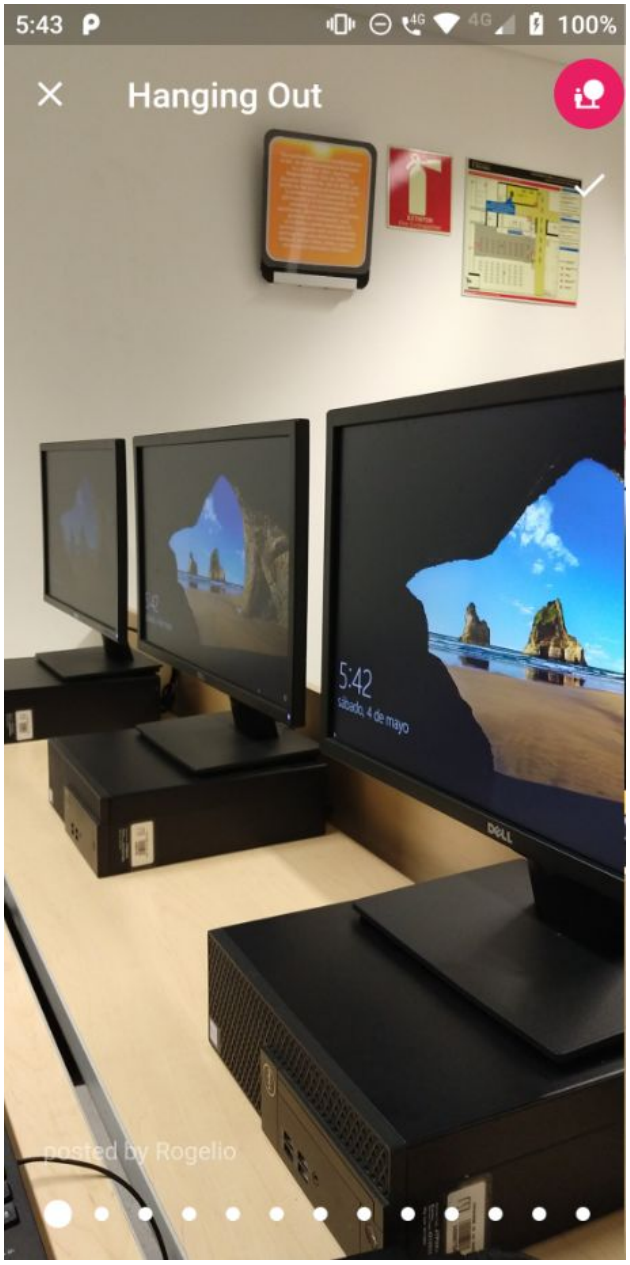
This refers to data stored inside the phone in an external folder, like the sd card.

In this case, data pictures taken by the user aren’t saved by default to the sd card, but the user can save his or any other picture after it has been successfully posted.

Opening an available picture yields the screen below (attention to the upper right corner).

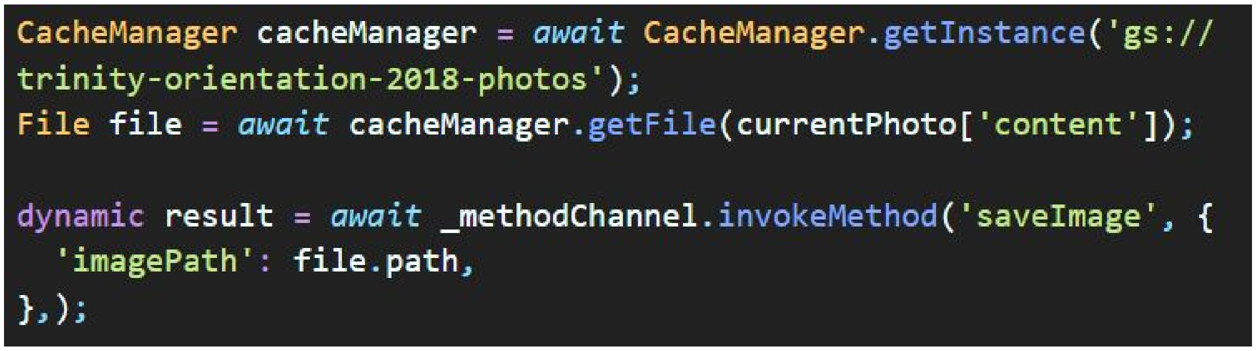


After hitting the save button on the upper right, a checked mark appears.



This, however, doesn’t actually save the image. If the user closes the Photo and opens it again, the save button appears again, as there is no logic behind showing or hiding the disquette icon in favour of a checked mark icon.

Take a look at what this is doing to save the image:



```
CacheManager cacheManager = await CacheManager.getInstance('gs://
trinity-orientation-2018-photos');
File file = await cacheManager.getFile(currentPhoto['content']);

dynamic result = await _methodChannel.invokeMethod('saveImage', {
  'imagePath': file.path,
},);
```

More on this method channel on the "code" report, since it has more to do with the way native code was implemented.

Lastly, external storage is also accessed when uploading a picture from the phone.

Remote Storage

Storage is largely managed with Firebase Storage, namely the schedule information, pictures and camera filters. Also, the app offers chatrooms, which are managed with Firebase Messaging.

Cache & Data Fetching Strategies

Cache is used to keep images in the app. Already loaded pictures are kept in the app for the user to see when they open the app. This avoids the need to fetch multimedia files from the server, and it's a permanent cache.

Cache is also used to keep camera filters readily available too. When a filter is loaded, it is cached. It's also a permanent cache.

To achieve this, the library flutter_cache_manager is used, and two files are implemented:

- cached_firebase_image.dart
- flutter_firebase_cache_manager

It's unclear why these libraries are implemented by the developer.

The fetching strategy used is pull-based. Only when the user asks for cached data is it downloaded and kept within the app. When the user first tries to load a filter a spinner is shown, and on airplane mode, a spinner is shown forever.

GIF 1: assets/images/performance/noLoadedFilters(1).gif

When the user has interacted with a filter it will show, but new ones won't load still.
GIF 2: assets/images/performance/loadedFilters(1).gif

UI, Rendering & Animations

For the UI we did guerilla testing to find out what was working well and what wasn't for both UIs. The questions were:

- Do you like this app?
- Particularly, do you like the colors chosen for this app?
- Was it hard to find your way through the interface?

We also took note on other comments and reactions by the user, and we asked which of the two apps they liked more.

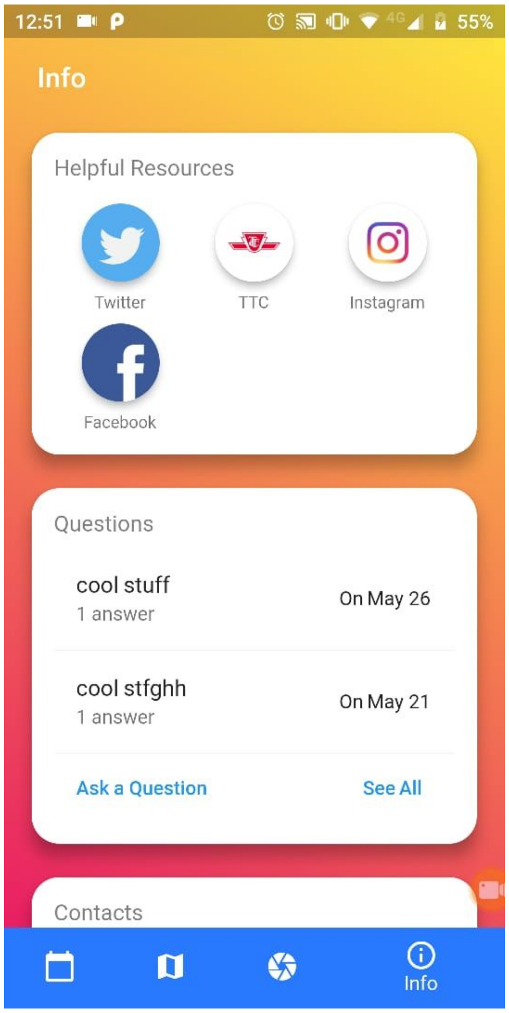
For rendering, we found when the interface wasn't responding as it should have and attempted to find out why.

For animations, we analyzed the most interesting ones for each app.

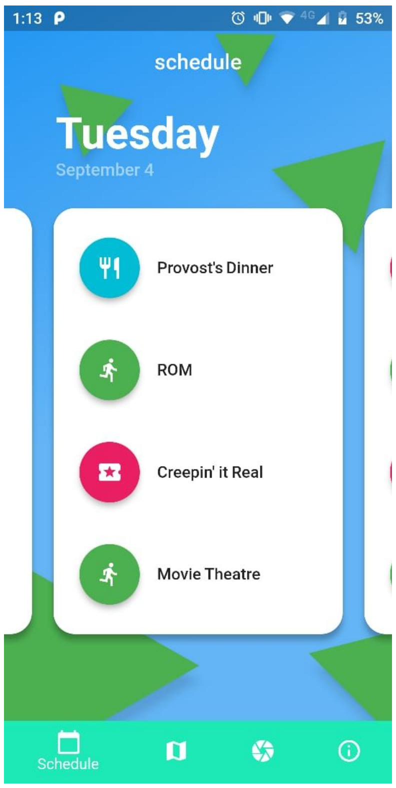
UI

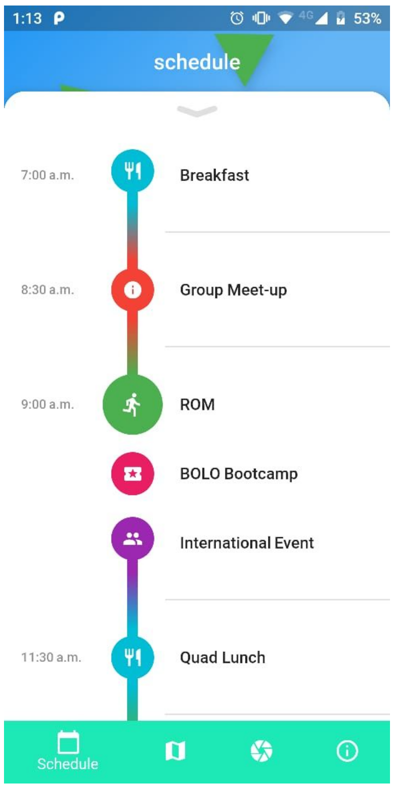
Do you like the look and feel of this app?
All of the interviewed people liked this app, mostly because of the animations.

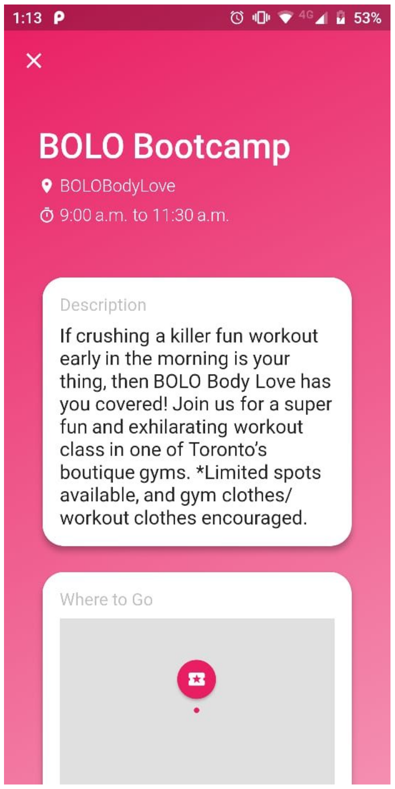
Particularly, do you like the colors chosen for this app?
Only three of interviewed people didn't like the colors, particularly on the 'Info' view, saying it was rather hard on the eyes. One said it was too 'Instagram-ish', which is true.



Was it hard to find your way through the interface?
People generally found their way through the app without issues, however, it took them a couple seconds to find out that on the schedule view, each card contained buttons, and each element in the bottom sheet led to another window.





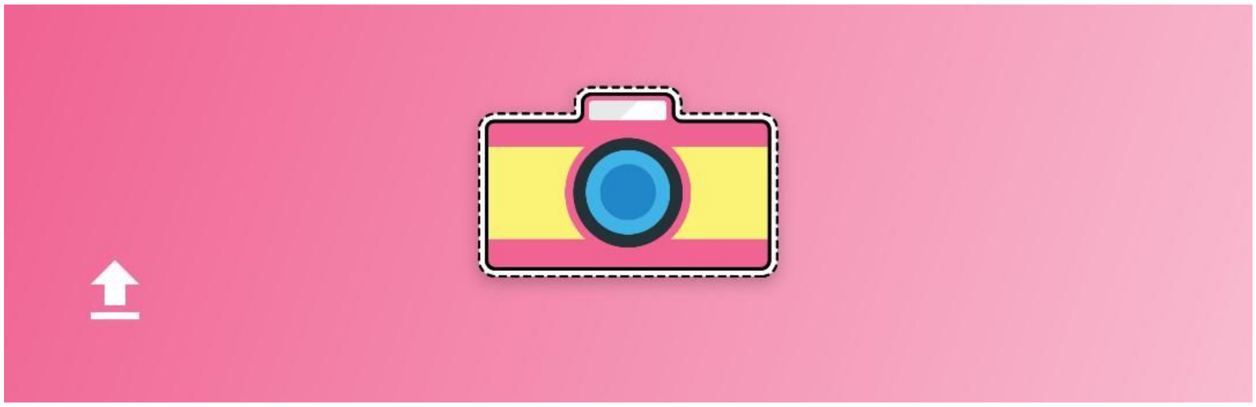


If crushing a killer fun workout early in the morning is your thing, then BOLO Body Love has you covered! Join us for a super fun and exhilarating workout class in one of Toronto's boutique gyms. *Limited spots available, and gym clothes/ workout clothes encouraged.

Further comments

A couple people commented on how fun it was to take pictures and see them displayed after a couple seconds on the app.

One person commented on the camera button, saying it was ugly. Can't say we disagree on that one.



People generally liked how the app was always fluid.

One person commented that it would be cool to do a similar app as a roadmap for the induction days in Los Andes.

Accessibility

We ran Google’s Accessibility Scanner and found the following suggestions.

Item labels
Accessibility Scanner suggests an item label for all clickable items. In 2T2000s, several Widgets have no label, like the social media buttons in the Info tab.

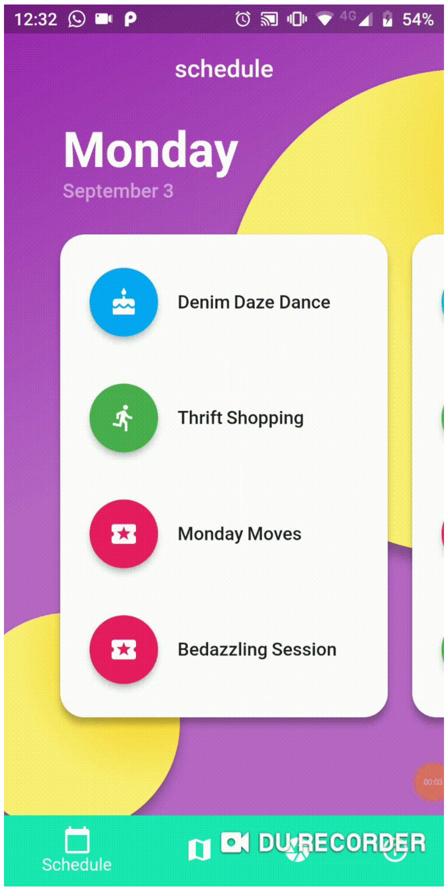
Color contrast
Color contrast was an issue in two screens.

Button size
Button size was a warning on two screens.

Rendering

This section is rather short since we couldn’t profile the use of the GPU from within the app, however, there is one situation when the interface yanks.

It occurs when the user drags from the first card to the second, on the schedule view.

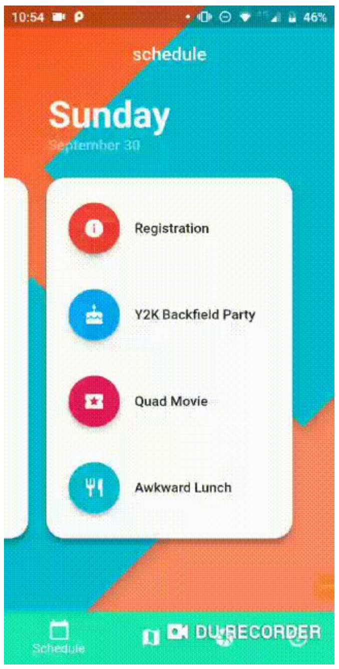


This can be fixed by switching to another physics handler. Currently, the app uses ScrollPhysics, a simple rename to AlwaysScrollableScrollPhysics would fix the issue. This scroll physics handler has no 'deactivated' timeframe when the user is unable to drag.

What is animated?

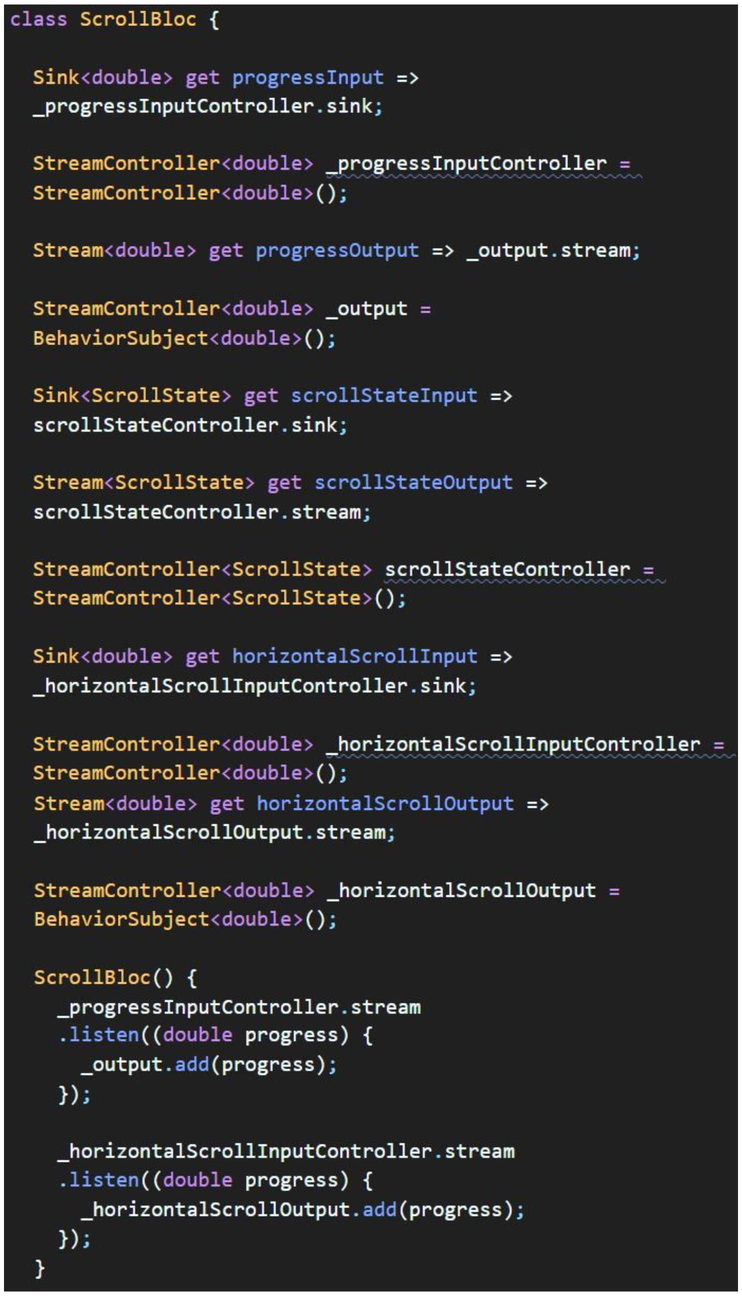
2T2000s:
Several parts of the app use custom animations, and great part of the code is dedicated to animating the app. The most prominent and obvious custom animation is the one found on the home screen.

Home screen animation



Animation1

On the ‘Code’ report, the BLoC pattern used is described in more detail 1 2. However, one of the two blocs used is only to manage scroll information. The bloc is used to define a feed of the scroll state in the form of a few controllers, sinks (data flows to the bloc) and streams (data flows to the widget).



```
class ScrollBloc {

  Sink<double> get progressInput =>
  _progressInputController.sink;

  StreamController<double> _progressInputController =
  StreamController<double>();

  Stream<double> get progressOutput => _output.stream;

  StreamController<double> _output =
  BehaviorSubject<double>();

  Sink<ScrollState> get scrollStateInput =>
  scrollStateController.sink;

  Stream<ScrollState> get scrollStateOutput =>
  scrollStateController.stream;

  StreamController<ScrollState> scrollStateController =
  StreamController<ScrollState>();

  Sink<double> get horizontalScrollInput =>
  _horizontalScrollInputController.sink;

  StreamController<double> _horizontalScrollInputController =
  StreamController<double>();
  Stream<double> get horizontalScrollOutput =>
  _horizontalScrollOutput.stream;

  StreamController<double> _horizontalScrollOutput =
  BehaviorSubject<double>();

  ScrollBloc() {
    _progressInputController.stream
    .listen((double progress) {
      _output.add(progress);
    });

    _horizontalScrollInputController.stream
    .listen((double progress) {
      _horizontalScrollOutput.add(progress);
    });
  }
```

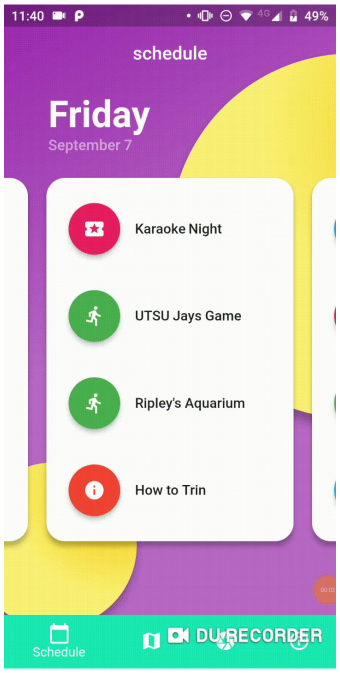
The state of the scroll is transformed eventually into a change in color and shape, which is what happens in the schedule view. To achieve this, several steps are made.

First, an AnimatedBackground class is defined. This receives a ‘t’ parameter, indicating the progress of the animation.

Second, the elements are defined. This includes an array of the background colors and the shapes, square, lines and triangles. In the example, two screens are shown, one with orange background with blue squares on top, and one with green background with six pink ‘lines’ on top.

Third, different functions are defined. They take the parameter ‘t’ and apply a transformation on the background color and the foreground shapes. There are four different functions, each defining how a screen will be animated. In this example, two of these functions are fired, interpolating between the two background colors, scaling and rotating from the middle of the screen and outwards.

Lastly, this animation implements its own Scrolling Physics, defined in ‘class MyScrollPhysics extends ScrollPhysics’. This overrides several methods, but the most interesting of them is the ‘carriedMomentum’ method. Defining a new formula for the momentum is what makes it possible to see this bouncy animation.

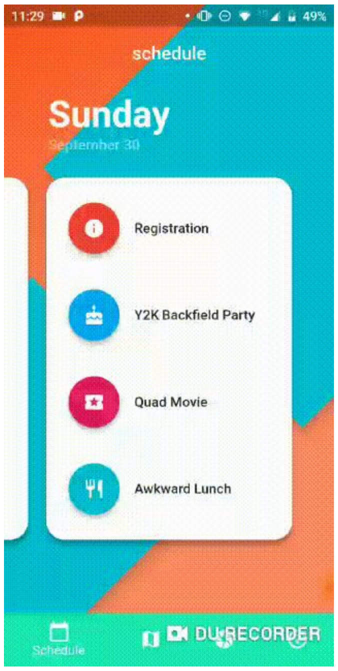


animation31

All in all, this is very complex in its implementation and the fact that it’s all implemented in a single file with over 10 different classes makes it extremely hard to understand. This animation alone is comprised of over 800 lines of codes. It’s also something unlike anything we’ve seen on an App.

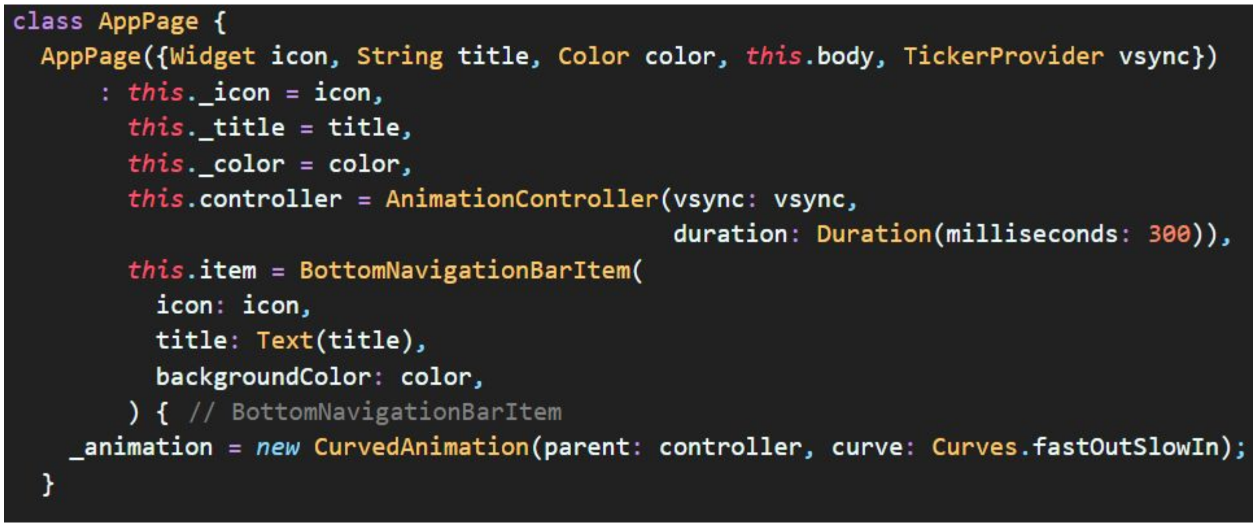
Bottom Navigation Bar Animation

This animation is more modest than the one in the home screen.



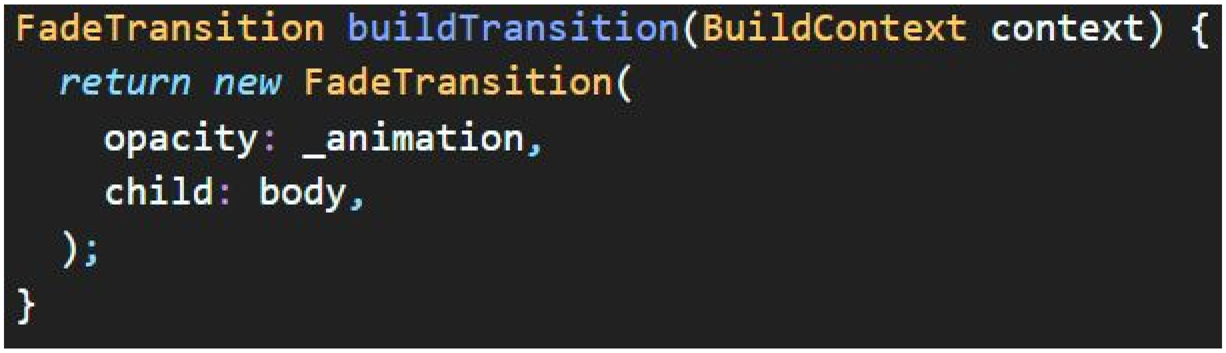
GIF DE ANIMATION2

It's built with an AnimationController set on a duration of 300 milliseconds. An Animation is built using CurvedAnimation on a fastOutSlowIn fashion.



```
class AppPage {
  AppPage({Widget icon, String title, Color color, this.body, TickerProvider vsync})
      : this._icon = icon,
        this._title = title,
        this._color = color,
        this.controller = AnimationController(vsync: vsync,
                                              duration: Duration(milliseconds: 300)),
        this.item = BottomNavigationBarItem(
          icon: icon,
          title: Text(title),
          backgroundColor: color,
        ) { // BottomNavigationBarItem
    _animation = new CurvedAnimation(parent: controller, curve: Curves.fastOutSlowIn);
  }
```

This animation is used on a FadeTransition that wraps each element in the BottomNavigationBar.



```
FadeTransition buildTransition(BuildContext context) {
  return new FadeTransition(
    opacity: _animation,
    child: body,
  );
}
```

Connectivity

In order to test this, we came up with scenarios, in order to reproduce them for all apps. Airplane mode was used to interrupt internet connection.

1. Install the app, turn on airplane mode and open the app. Try to interact with it.
2. Interact with the app, exit, turn on airplane mode and open the app. Try to interact with it.
3. Interact with the app, but before using each feature, turn on airplane mode.
4. Interact with the app, turning on airplane mode in the middle of executing each feature.

We will also cover the feedback that the user receives when there is no internet connection. Is it blocking? Is the error message unexpressive? Let’s find out!

Interactions

2T2000s:
The features chosen to analyze are:
Puede ser gif

SpaceX:
The features chosen to analyze are:
Puede ser gif

Scenario 1

Scenario 2

Scenario 3

Scenario 4

Connectivity Antipatterns

The app loses content when it’s used without internet connection. The Lost Content issue is described as a situation when:

- An app does not have connectivity
- It shows empty, incomplete, or blurred content where it is supposed to be
- The app does not report anything about a connectivity problem

User Content type. It is lost when a picture is taken without internet, it isn’t saved anywhere so it can be sent afterwards, when there is internet.

Blank Map type also happens.

Lastly, Blank Image type happens as well.

Conclusions

Code

This section describes the most prominent design/architecture patterns implemented, and which libraries were used to achieve which features. Also, we take a closer look at Native code for both Android and iOS. 2T2000s does a very good job at implementing a pattern created and recommended by Google for Flutter, while SpaceX GO! is more orthodox and uses scoped models.

Patterns

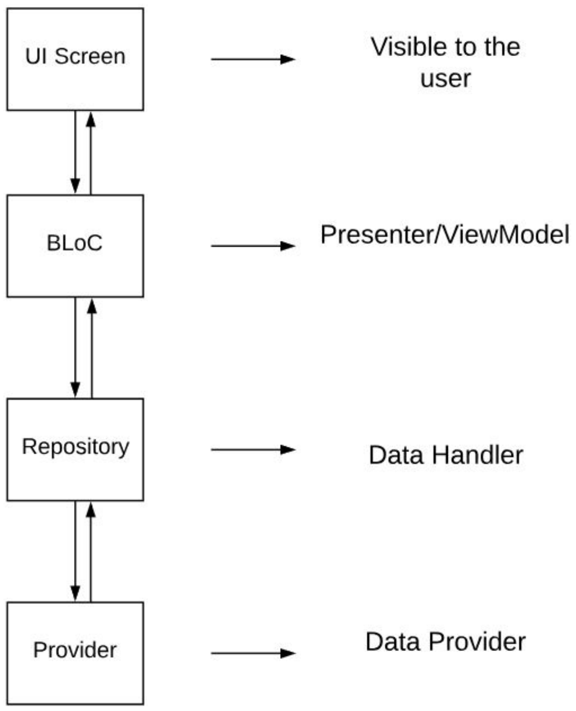
2T2000S

This app implements the BLoC pattern. BLoC stands for Business Logic Component and is a pattern created and recommended by Google for Flutter.

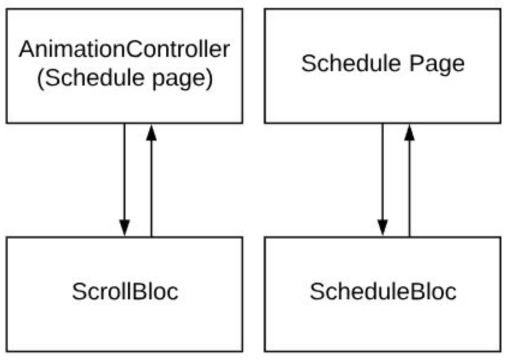
This pattern is rather complicated to wrap your head around if it's the first time you use Flutter. For us, it was easier to start with other options, like passing around Widget data from parent to child, and vice versa, before switching to using Scoped Models. There are several options to handling state, but BloC greatly reduces the complexity of listening to changes in the data.

BloC makes great use of reactive programming and the observer pattern by using the RxDart library, and streams of data by using the StreamBuilder widget, as well as Object Models.

The way this pattern is actually implemented isn't clear, however. The pattern looks as follows.

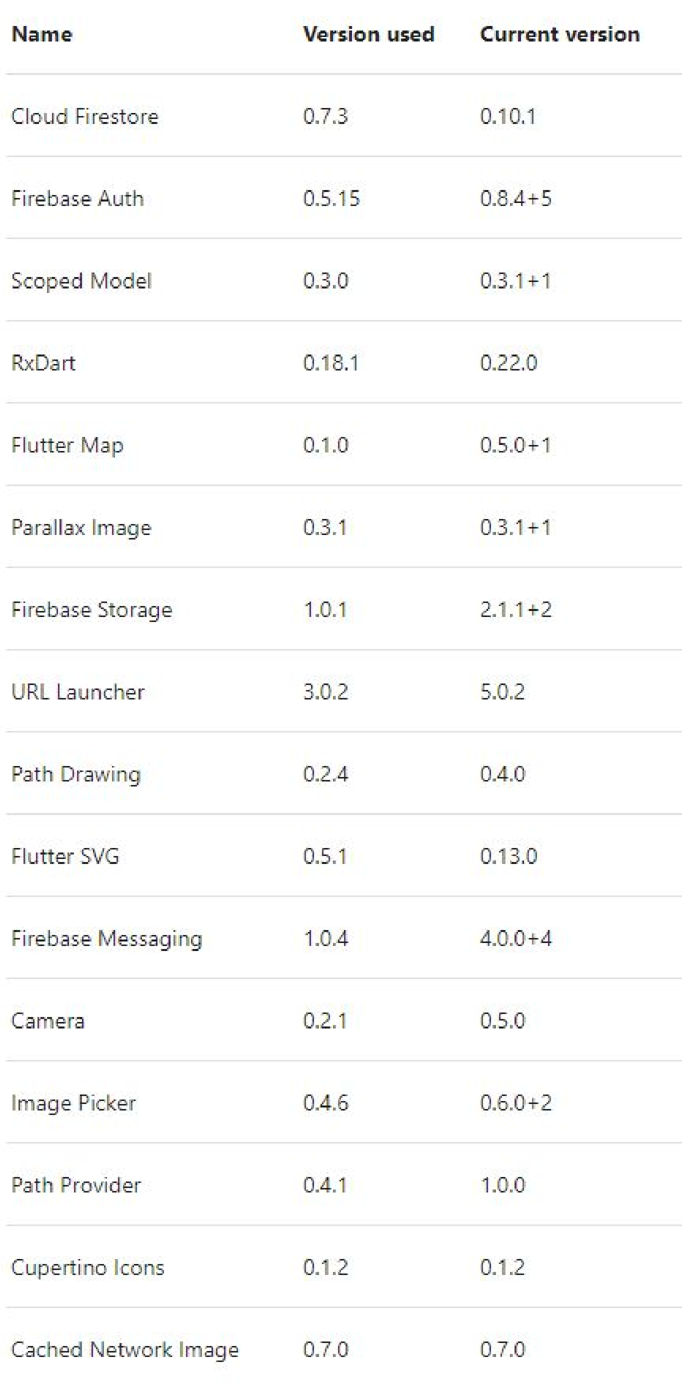


The difference is that there is no repository in this project. It's implemented as follows.



Libraries

Libraries declared by 2T2000S:



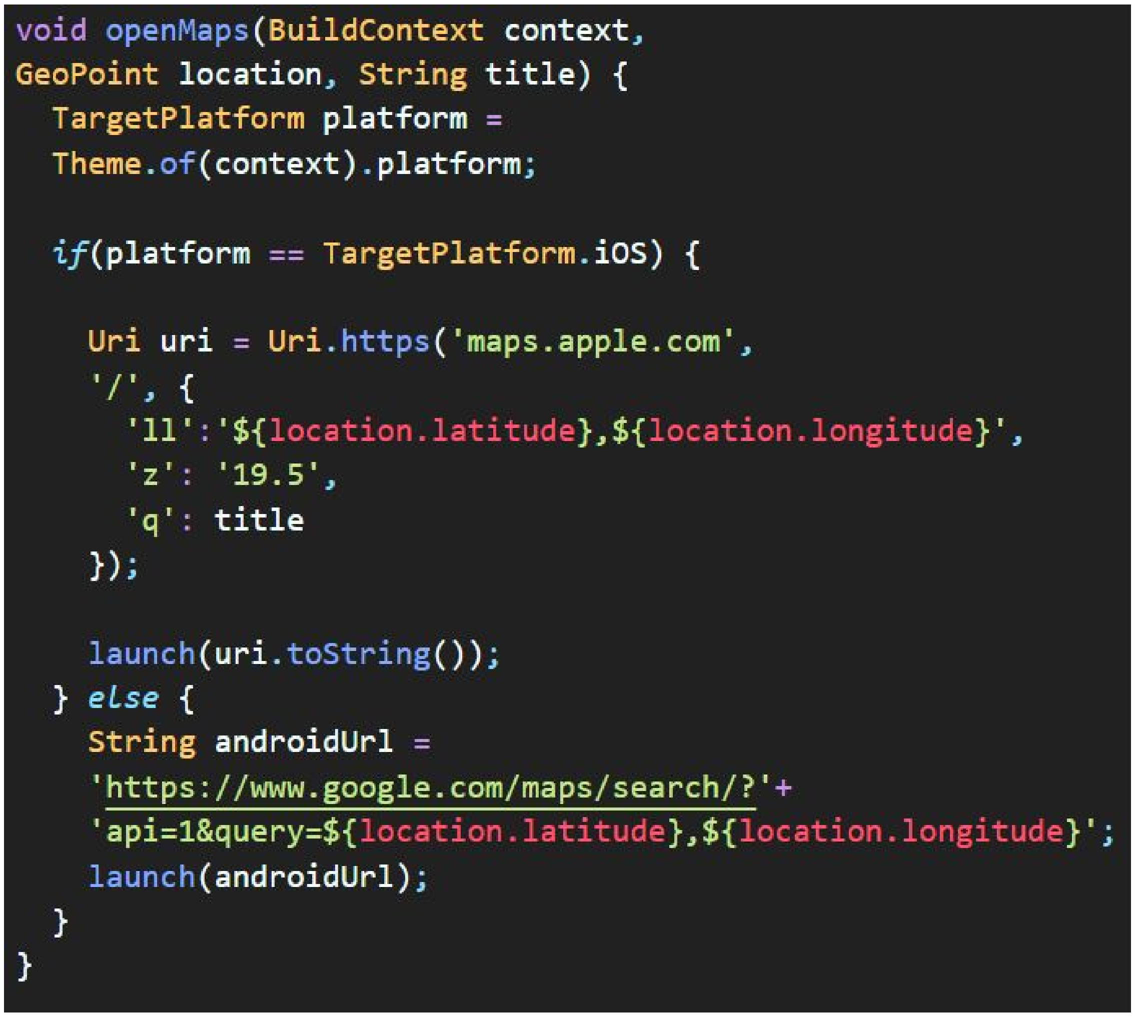
Name	Version used	Current version
Cloud Firestore	0.7.3	0.10.1
Firebase Auth	0.5.15	0.8.4+5
Scoped Model	0.3.0	0.3.1+1
RxDart	0.18.1	0.22.0
Flutter Map	0.1.0	0.5.0+1
Parallax Image	0.3.1	0.3.1+1
Firebase Storage	1.0.1	2.1.1+2
URL Launcher	3.0.2	5.0.2
Path Drawing	0.2.4	0.4.0
Flutter SVG	0.5.1	0.13.0
Firebase Messaging	1.0.4	4.0.0+4
Camera	0.2.1	0.5.0
Image Picker	0.4.6	0.6.0+2
Path Provider	0.4.1	1.0.0
Cupertino Icons	0.1.2	0.1.2
Cached Network Image	0.7.0	0.7.0

Libraries declared by SpaceX-Go:

Name	Version used	Current version
Flutter Mailer	0.4.0+1	0.4.1
Flutter Web Browser	0.11.0	1.1.3
Dio	2.1.2	2.1.3
Cached Network Image	0.7.0	0.7.0
Shared Preferences	0.5.2	0.5.2
Add 2 Calendar	1.2.2	1.2.2
Flutter Local Notifications	0.6.1	0.7.0
Flutter Map	0.4.0	0.5.12
Flutter Swiper	1.1.6	1.1.6
Material Search	0.2.8	0.2.8
Sliver Fab	1.0.0	1.0.0
Flutter i18n	0.6.3	0.6.3
Flutter i18n	0.6.3	0.6.3
Intl	0.15.7	0.15.8
Package info	0.4.0+3	0.4.0+3
Quick Actions	0.3.0+1	0.4.0+3
Scoped Model	1.0.1	1.0.1
Share	0.6.1	0.6.1+1
System Setting	0.1.2	1.0.6
Font Awesome Flutter	8.4.0	8.4.0

Platform Specific Code

The app runs platform specific code on one situation only. On either the map view or the event view, when the user taps on the map, the link opened is different for each platform. iOS directs to maps.apple.com and Android directs to Google Maps.



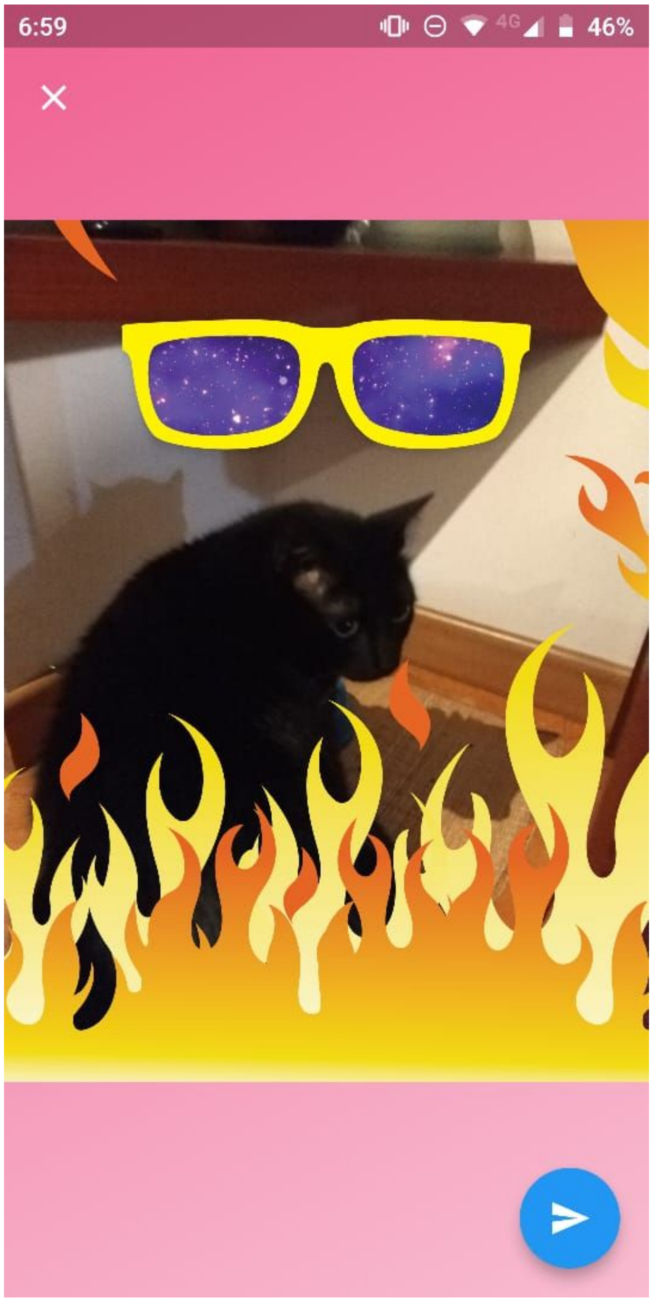
```dart
void openMaps(BuildContext context,
GeoPoint location, String title) {
  TargetPlatform platform =
  Theme.of(context).platform;

  if(platform == TargetPlatform.iOS) {

    Uri uri = Uri.https('maps.apple.com',
    '/', {
      'll':'${location.latitude},${location.longitude}',
      'z': '19.5',
      'q': title
    });

    launch(uri.toString());
  } else {
    String androidUrl =
    'https://www.google.com/maps/search/?'+
    'api=1&query=${location.latitude},${location.longitude}';
    launch(androidUrl);
  }
}
```

Native Code

The project makes two channels. Channels are the way that Flutter can communicate with native functions, that can be either already part of Android or iOS, or custom made. In this app, its the latter.

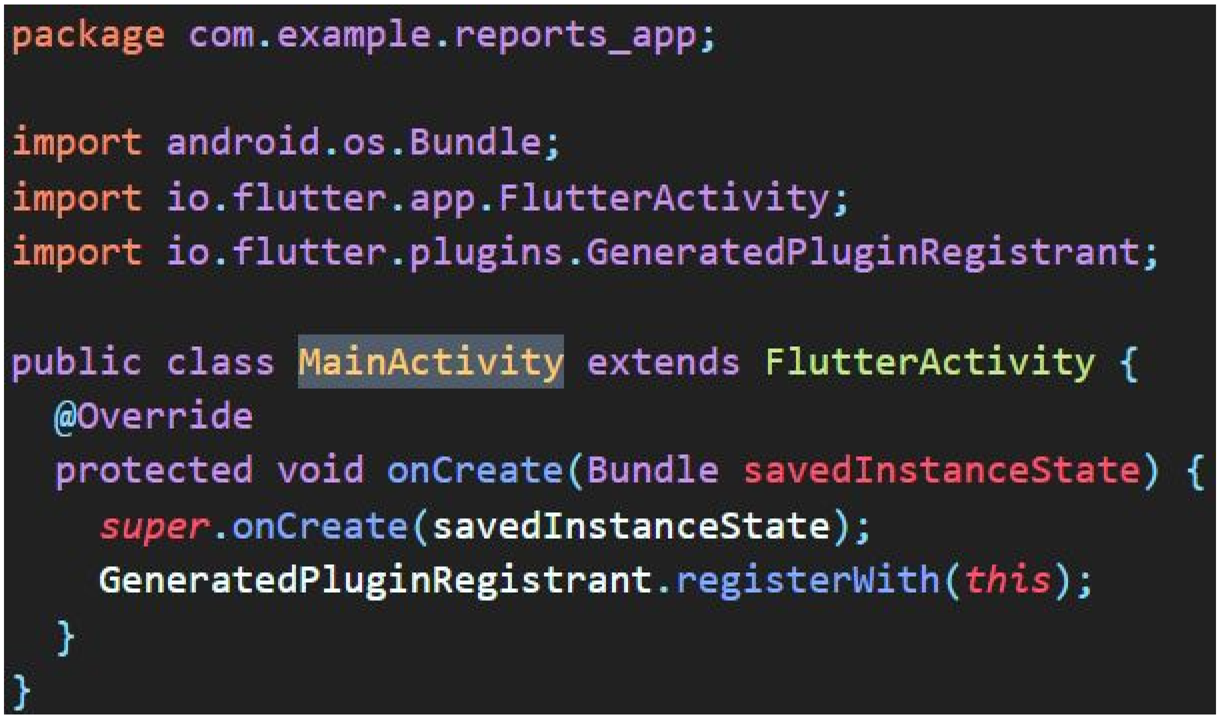
This app implements a channel to save an image to the phone and a channel to add the image filters. The first case sounds like an excellent candidate for a plugin, right? Well, a plugin has just been published a few days ago. On the overlay front, however, there is no option to actually combine two pictures into one, which is the case of the filters offered.

This is what the filters look like



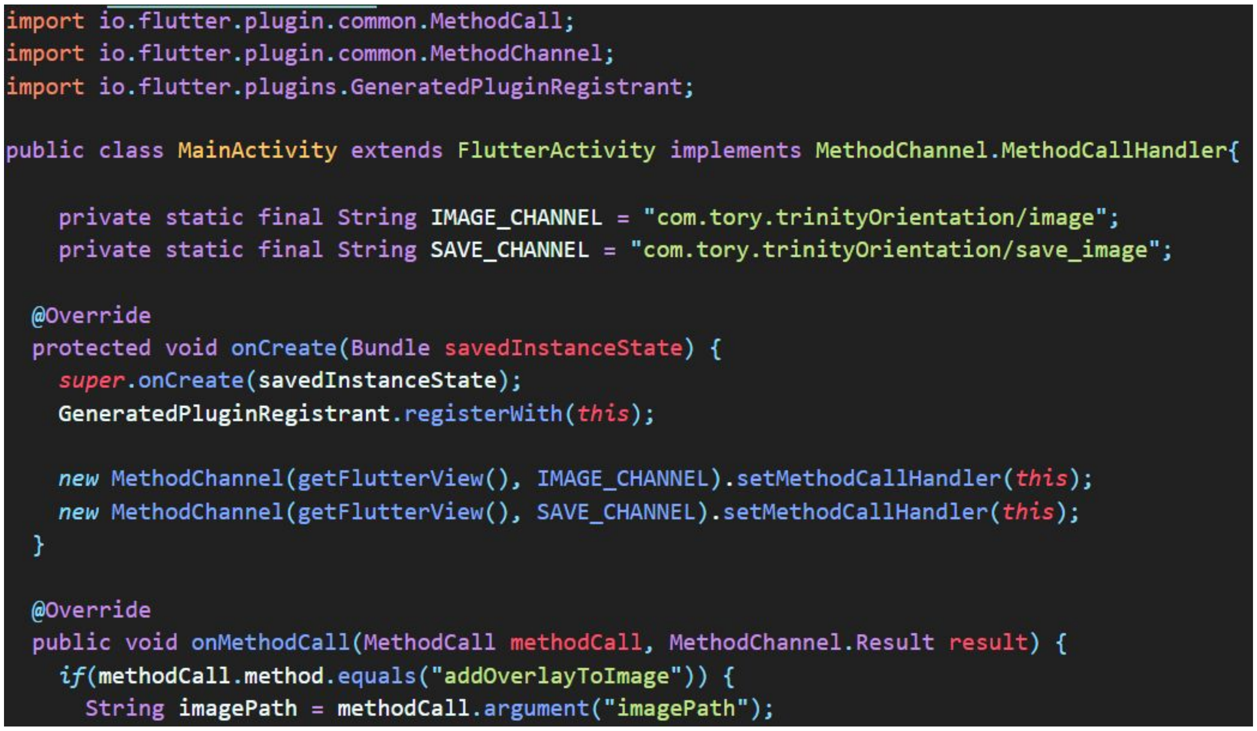
Native Android Code

A default MainActivity.java file in a Flutter app looks like this



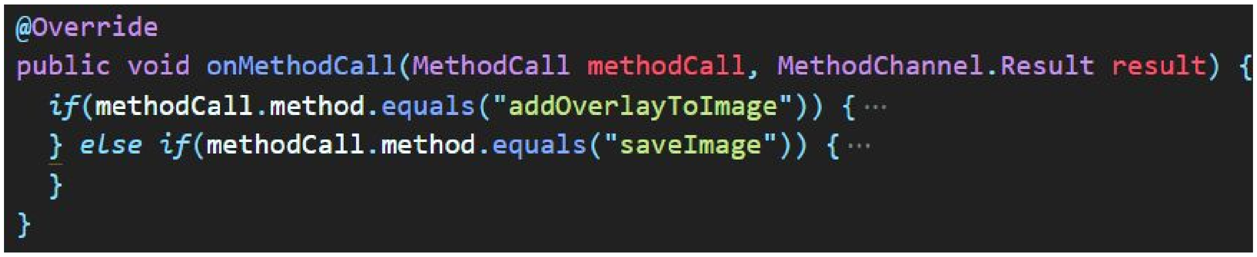
```java
package com.example.reports_app;

import android.os.Bundle;
import io.flutter.app.FlutterActivity;
import io.flutter.plugins.GeneratedPluginRegistrant;

public class MainActivity extends FlutterActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    GeneratedPluginRegistrant.registerWith(this);
  }
}
```

And this is what MainActivity.java on this app looks like



```java
import io.flutter.plugin.common.MethodCall;
import io.flutter.plugin.common.MethodChannel;
import io.flutter.plugins.GeneratedPluginRegistrant;

public class MainActivity extends FlutterActivity implements MethodChannel.MethodCallHandler{

    private static final String IMAGE_CHANNEL = "com.tory.trinityOrientation/image";
    private static final String SAVE_CHANNEL = "com.tory.trinityOrientation/save_image";

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    GeneratedPluginRegistrant.registerWith(this);

    new MethodChannel(getFlutterView(), IMAGE_CHANNEL).setMethodCallHandler(this);
    new MethodChannel(getFlutterView(), SAVE_CHANNEL).setMethodCallHandler(this);
  }

  @Override
  public void onMethodCall(MethodCall methodCall, MethodChannel.Result result) {
    if(methodCall.method.equals("addOverlayToImage")) {
      String imagePath = methodCall.argument("imagePath");
```

The two channels end up calling onMethodCall, so it has to handle them



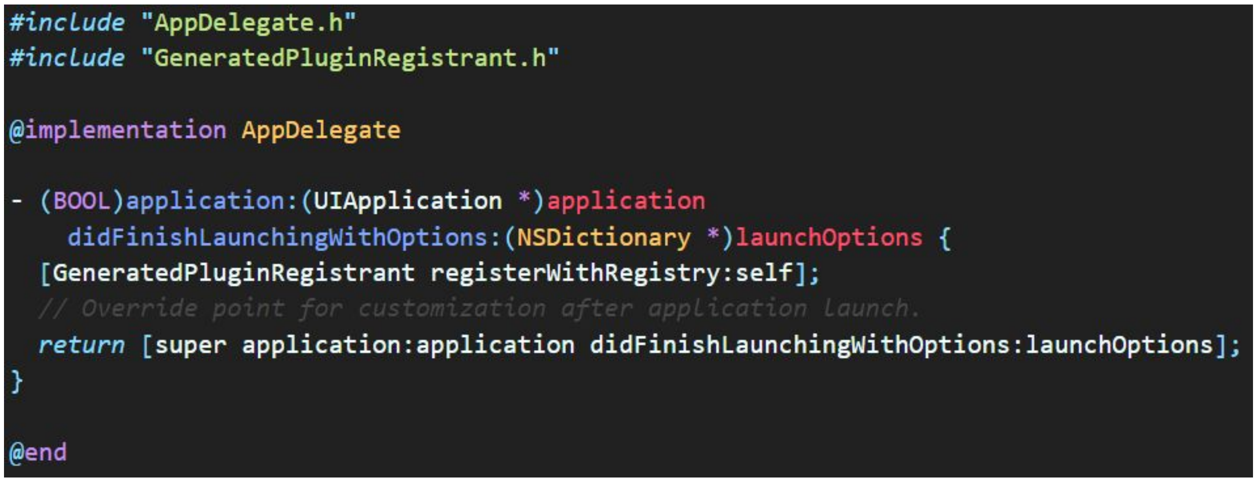
```
@Override
public void onMethodCall(MethodCall methodCall, MethodChannel.Result result) {
  if(methodCall.method.equals("addOverlayToImage")) {···
  } else if(methodCall.method.equals("saveImage")) {···
  }
}
```

On the ‘addOverlayToImage’ call, things like phone orientation need to be handled to apply the overlay correctly, and the final image is comprised of a canvas with two Bitmaps drawn onto it.

On the ‘saveImage’ call, the image is never actually saved or written into a file, which is presumably why it doesn’t actually save the image.

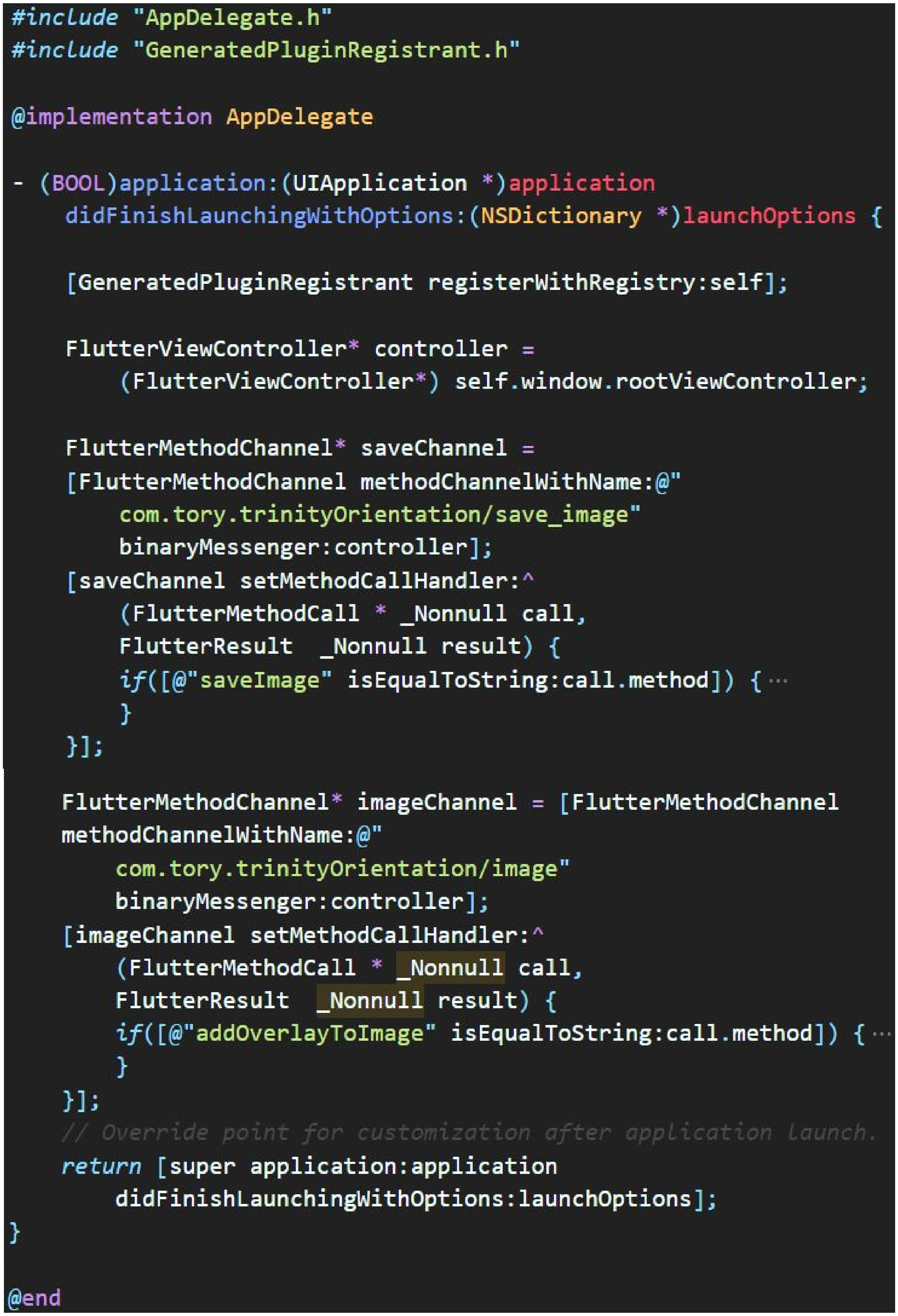
Native iOS Code

This was implemented in Objective-C on the AppDelegate.m file. A default AppDelegate.m file on Flutter looks like this



```
#include "AppDelegate.h"
#include "GeneratedPluginRegistrant.h"

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
  [GeneratedPluginRegistrant registerWithRegistry:self];
  // Override point for customization after application launch.
  return [super application:application didFinishLaunchingWithOptions:launchOptions];
}

@end
```

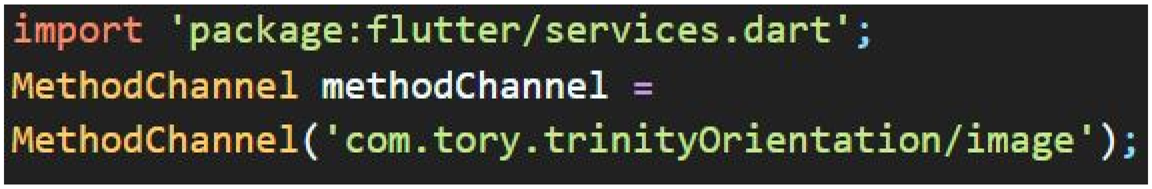
On the app, a function was implemented in pretty much the same way it was in Java, but on Objective-C, with the exception that in iOS, the orientation isn’t handled directly as it is in Android, but comparing the height and width of the screen.



```
#include "AppDelegate.h"
#include "GeneratedPluginRegistrant.h"

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    [GeneratedPluginRegistrant registerWithRegistry:self];

    FlutterViewController* controller =
        (FlutterViewController*) self.window.rootViewController;

    FlutterMethodChannel* saveChannel =
    [FlutterMethodChannel methodChannelWithName:@"
        com.tory.trinityOrientation/save_image"
        binaryMessenger:controller];
    [saveChannel setMethodCallHandler:^
        (FlutterMethodCall * _Nonnull call,
        FlutterResult  _Nonnull result) {
        if([@"saveImage" isEqualToString:call.method]) {···
        }
    }];

    FlutterMethodChannel* imageChannel = [FlutterMethodChannel
    methodChannelWithName:@"
        com.tory.trinityOrientation/image"
        binaryMessenger:controller];
    [imageChannel setMethodCallHandler:^
        (FlutterMethodCall * _Nonnull call,
        FlutterResult  _Nonnull result) {
        if([@"addOverlayToImage" isEqualToString:call.method]) {···
        }
    }];
    // Override point for customization after application launch.
    return [super application:application
        didFinishLaunchingWithOptions:launchOptions];
}

@end
```

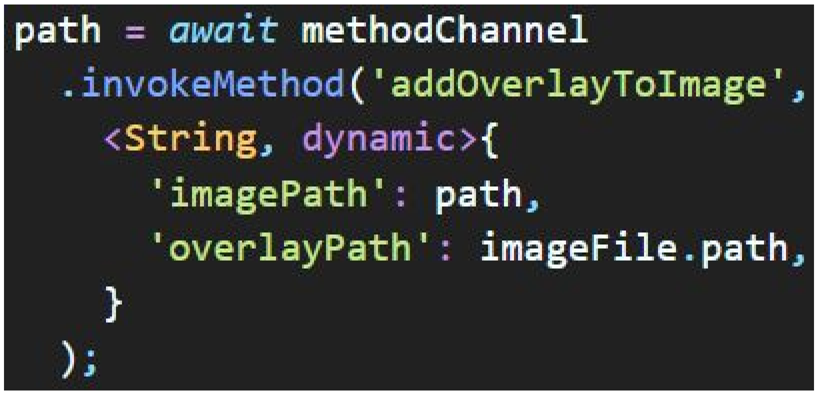
Running Native Code

After the desired channels have been defined, running native code from Flutter is simple. First, a MethodChannel must be declared and initialized. Method channel is part of the Flutter/services package for Dart.



```dart
import 'package:flutter/services.dart';
MethodChannel methodChannel =
MethodChannel('com.tory.trinityOrientation/image');
```

It is used with invokeMethod, and Flutter will decide whether to invoke Android's channel on MainActivity.java or iOS' channel on AppDelegate.m.



```dart
path = await methodChannel
  .invokeMethod('addOverlayToImage',
    <String, dynamic>{
      'imagePath': path,
      'overlayPath': imageFile.path,
    }
  );
```

Good Programming Practices

BLoC architectural pattern. It's described as a good practice that focuses on the principle of detaching UI and State and Data Handling, and it is applied according, mostly, to the guidelines available.

All images shown in the app are cached using the cached_network_image library.

Bad Programming Practices

A BLoC antipattern was encountered. Streams are opened but are never closed or otherwise disposed of. This is listed as an antipattern for BLoC, but it's just generally a bad idea; it was also one of the points highlighted as a warning during the static code analysis, in the performance report.

Maintainability is an issue in this project. This was done by a single person, and given the ‘one use only’ nature of the project, one could argue that maintainability is not so much of an issue, however, this project became a referent for how powerful Flutter can be to build beautiful and smooth interfaces and animations.

Specifically, code could be heavily rearranged. The project comprises of a bloc folder containing two bloc models and one data provider, an (unused) models folder, an implementation of a library in the firebase_cache library (we couldn’t figure why it was implemented, it’s just the regular library files), 12 loose UI files and 1 utils file.

Each of the UI files contains several (up to 25) classes. Granted, all of them belong to the same view, but a general good practice is to, when possible, limit a .dart file to one class, when it’s a StatelessWidget, or two, when it’s a StatefulWidget.