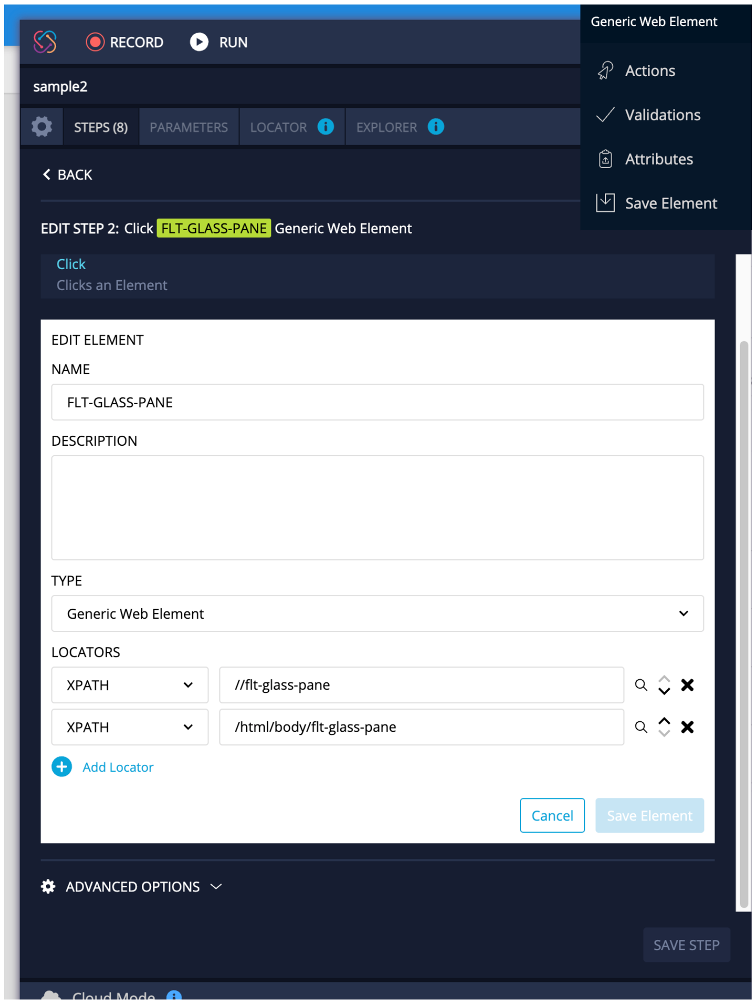
Flutter Integration Tests for Web Apps

TestProject didn't work for Desktop Flutter WebApps

We tried to use TestProject. (It is a now-dead platform that was based on Appium)

With Flutter, you could set your WebApp to render in HTML instead of CanvasKit (think Skia) but our architect really didn't want that for the target app being tested. So, with CanvasKit, all TestProject saw was a "flt-glass-pane" element. Just a pane that is painted as part of this CanvasKit. So, "nothing to see here".



EDIT STEP 2: Click FLT-GLASS-PANE Generic Web Element

Click
Clicks an Element

EDIT ELEMENT

NAME: FLT-GLASS-PANE

DESCRIPTION:

TYPE: Generic Web Element

LOCATORS
XPATH //flt-glass-pane
XPATH /html/body/flt-glass-pane

So what's wrong in the depths of Appium? Well, Appium is rather mobile device focused and they seem to have never thought of supporting desktop browsers. And it doesn't look hopeful.
https://github.com/appium-userland/appium-flutter-driver/issues/263
Could it work for testing in a mobile device browser? Maybe. The target of our tests was an application that took up a lot of desktop screen real estate; we gave up on TestProject/Appium.

Flutter Integration Tests

Brace yourself, "They are not fully as advertised; can't debug them, can't put them in groups, one test per dart file. It seems a bit like not many people are writing these tests and they work just enough that Google can say they can be done but the documentation isn't so clear."

Start by trying to follow this really good tutorial:

https://www.raywenderlich.com/29321816-integration-testing-in-flutter-getting-started
https://docs.flutter.dev/cookbook/testing/integration/introduction

Dos and Don'ts

Ensure the target app main compiles and runs from vscode.
Ensure all SSH key issues are resolved so that the app can be built when running "flutter drive".

You have to install the right ChromeDriver for the version of Chrome Browser you are using.
https://chromedriver.chromium.org/downloads

Watch in the Chromedriver shell because sometimes Google automatically updates your version of Chrome and you'll see something like this "[WARNING]: This version of ChromeDriver has not been tested with Chrome version 104." which is your CLUE that you are running the WRONG version of Chromedriver!

WINDOWS

```
flutter drive --driver=test_driver/integration_test.dart
--target=integration_test/Login_Logout.dart -d chrome | ./integration_test/testout.sh
```

*** note the "-d chrome" (This isn't used on OSX, it seems to be needed on windows)

OSX

```
flutter drive --driver=test_driver/integration_test.dart
--target=integration_test/Login_Logout.dart | ./integration_test/testout.sh
```

*** It seems tests optionally can be run without the "-d chrome" option. Why? I don't know, it somehow defaults to chrome which it doesn't on Windows.

Problems

The APP starts but the test doesn't fill in any fields?
- You probably forgot to run chromedriver --port=4444 in a shell

The APP starts but the test doesn't fill in any fields?
- Does your version of Chrome match the version of chromedriver? Check the chromedriver shell and see if it is complaining.

Writing Test Cases

How did we end up using Flutter Integration Test, one test per dart file?

- Normally "group" is used to wrap a series of Flutter Integration tests ("testWidgets" calls) but through trial and error, it was determined that this didn't work properly for Web App testing.
- Why not a series of "testWidgets" in one dart file and only load the app once? That would be efficient, but sadly it just didn’t work. There were attempts to reset the app but they failed. In the end, we were stuck with one test per dart file. Each one starts with a new image of the web app.

Trying to understand the results

Reality:
- Flutter driver results are printed to console
- When you run a testWidget you get the final result and exception traceback dumps
- We need the Pass/Fail on each step.

Quick solution:
- We need to add a "Before step” statement for each step we were about to run because we could die while running the step and not fully know where we died. So, there are print statements added with coded labels.
- A shell script was written (testout.sh) to generate clear Pass/Fail info and to try to pick out an error summary. The full console test results end up in output.txt.

Handling exceptions

- Sadly, there are no exception types available, they are just strings. All we can do is parse strings. But the strings are completely inconsistent.
- Our code was set up to pass an exception call back. When the exception happens, your call back is called and you can then look at the string but it is basically useless. So, in the end, it is a matter of ignoring exceptions on some calls or not.

Result console prints
- The console print statements are "wrapped" within our framework code; trying to hide/keep the end test cases clean.

Using a page object pattern

- There is a solid history for this in test automation (https://martinfowler.com/bliki/PageObject.html)
- The goal is to isolate the test cases from changes to fields/properties on each page. When changes happen, you change the "page" code and should not have to change each test case.
- But our app doesn't exactly have Pages. We really have "Panes"; you drag a field onto a pane. Each of those field types results in a different "Details" pane where you can assign labels, validations, etc for the field.
- Generally, there is a desire for the "expect" statements to be in the final test case and not the "page"/"pane" code. But, I have added "verify" methods for "panes", so that all fields on a pane could be checked; this would be used for-say login fields before login and after logout. Making sure nothing was missing.

Keys

- Keys are special internal code labels for fields/widgets within the app code.
- Anecdotally, Keys seem to result in better execution times. Finding a field by say ToolTip and then searching Ancestors and Descendants within the Widget tree seems to take a lot of time. What could take 3 minutes with a Key, ends up taking 20 minutes!

Passing Parameters to tests

- The wrapper/framework gets the user id and password for tests from environment variables but has defaults as well. This allows us to run tests in parallel using different accounts.
- There are other methods that could be used for this but this seemed simplest for now.

Running tests & CI/CD goals

- With Web App Integration tests, in theory, these could be run anywhere. We don't need a test farm because it's just a web app. You could set up a Docker image, and run tests there.
- I have not properly researched the processes/execution scripting/options to kick off the running test cases on mass, collect execution results, and control which tests have to be run in series (say building a form and then testing the form on an android device)
- I have configured a Windows box for the remote execution of tests.

When writing test cases gets paranormal
- There are "Things you can't see but are still there". I have run into a situation where a different field "Details" pane was being displayed, but the old pane fields were somehow still in part of the tree. So, a search found a field that wasn't visible.
- There are "Things you can see but aren't visible in the widget tree". This seems more impossible but I have had difficulties finding fields which are clearly visible. This may be a bug somewhere but it is an observation.

Getting testout.sh running

Windows
- The testout.sh script uses pcregrep. Install PCRE1 from
http://www.rexegg.com/pcregrep-pcretest.html

Run in a BASH shell (run 'bash' instead of 'cmd')

```
flutter drive --driver=test_driver/integration_test.dart
--target=integration_test/Login_Logout.dart -d chrome | ./integration_test/testout.sh
```

*** note the "-d chrome" (This isn't used on OSX, all I know that it is needed on windows)
*** testout.sh had to be modified compared to the OSX version; outputting results to output-temp.txt | tr -d '\000' > output.txt. Why? Some null that keeps messing with the result.

OSX

- It would seem I installed pcregrep version 1 through homebrew (pcre).
https://formulae.brew.sh/formula/pcre

Tests optionally can be run without the "-d chrome" option. Why? I don't know, it somehow defaults to chrome which it doesn't on Windows; trial and error (in my configuration).

```
flutter drive --driver=test_driver/integration_test.dart
--target=integration_test/Login_Logout.dart | ./integration_test/fdout.sh
```

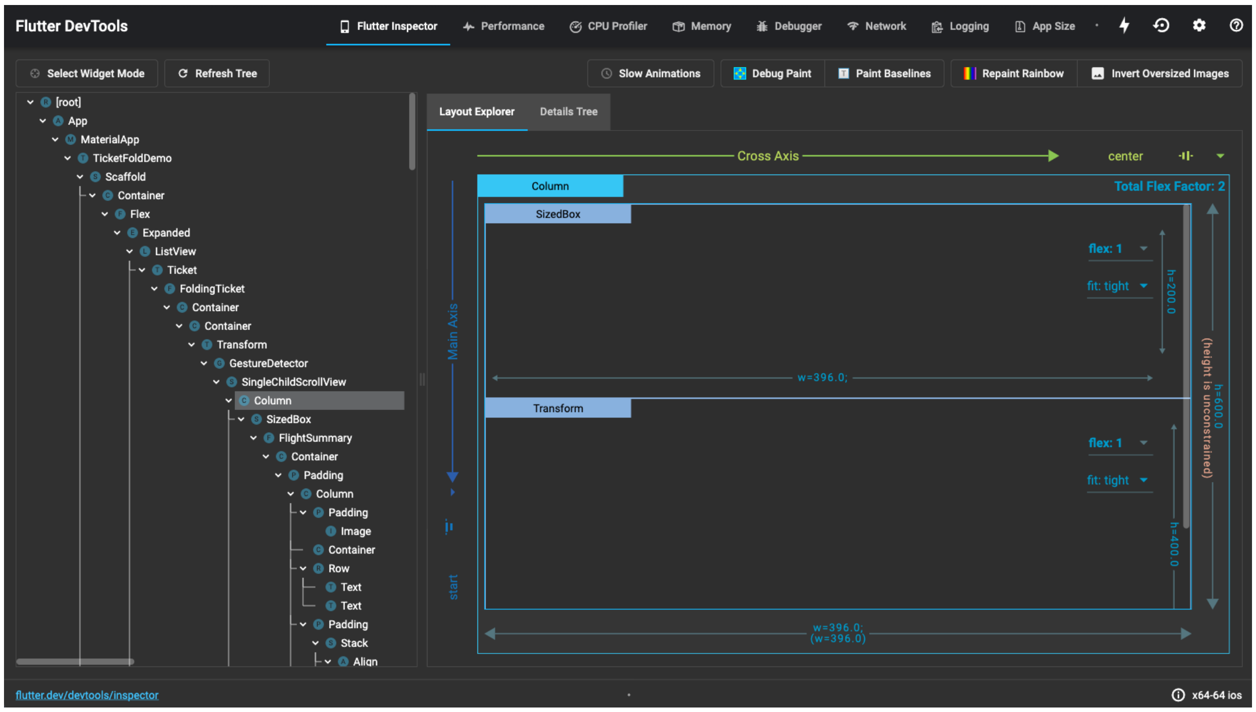
Why not Gherkin / Cucumber?

- Glad you asked! I guess it came down to following the “normal” tutorials; trying to get something working quickly. Then the problem was running into too many things that “should have worked”.Then adding to what was working, without “going the next step” of getting Cucumber working.

Ok, the gherkin package 2.0.0 says it still doesn’t support null-safety, that was another problem…there are other packages
https://pub.dev/packages/flutter_gherkin
https://github.com/jonsamwell/flutter_gherkin/blob/master/test/flutter_configuration_test.dart

Widget tree pain

- Yes, we are talking about “Flutter Inspector” and hunting through the widget tree to find a Key or what to search on to find the desired widget/data.
- Do remember to click the “Details Tree” tab. No, there are no search abilities.
- All you can do is use “Select Widget Mode” and click on the target field/widget. Yes, I have seen the app exit when you click on the field/widget; if this happens, you have to manually hunt through the tree instead.



- Why can’t I copy/paste key values from the widget explorer? Because you can’t. I did install an extension that half worked. It should have also allowed copy but it didn’t. It did successfully allow you to pass the selected text to another app. So, on OSX, it would load TextEdit with the text, then I could copy the key.

GitHub Files

The sample code tests a web app login screen. The testing code is there but the actual app that throws up the login fields and does authentication is not there. Yes, main.dart is just HelloWorld. The integration_test/lib/test_lib_common.dart file, the test_login_pane.dart and 3 sample test cases should be most interesting. The goals were to use panes and make the top-level test code as simple as possible. Yes, long-term or even short-term, we should have used Cucumber but it didn’t happen.

Manual Testing

If you are looking for a not-so-simplistic test spreadsheet (Google Sheets) to store manual test cases and then reference them as you record test results in those same sheets, check out my other post.