Code Testing Plan and Results

Humane Transport

Date last edited: Apr 1, 2021

Team Members:
Kelly Holtzman, Sana Khan, Mansi Patel, Clark Inocalla

—

The definition of software tests that we use

We use the definitions of unit, integration, and system software tests as described in the Atlassian continuous integration/continuous development (CI/CD) article by Pittet (2021): in short, we consider and use the types at separate levels of testing and not interchangeably. The Flutter framework considers system and integration tests to be the same.

For our purposes, we call anything that tests two or more Flutter widgets or classes, together, an integration test (even if it needs the entire running application to work); we call anything that's written to implement our system test procedures a system test. System tests generally involve more widgets/classes than an integration test and individually target an experience in the application: for example, starting a new Animal Transport Record when there is no internet connection.

—

Unit and integration test plan (in general, see application code directly)

The reader may find further identification, instructions, and discussion on Flutter tests and best practices using the Flutter online cookbook or manual (Flutter, 2021).

Most of our individual widgets and classes have unit tests. Any missing test cases are mentioned in the Future Work section. We use the previous definition of unit tests and say that each function of a class or widget has at least one corresponding unit test that exercises its “success” case or the general case when input information has defined output.

More important classes that support integral operations of the server portion of our application have more unit tests per function and integration tests, exercising cases of success and failure behaviour (for example, `animal_transport_record_editing_screen_test.dart` is one such integration test grouping the individual forms of the Animal Transport Record).

We define failure behaviour as what the function returns or does in the case it cannot perform what it has been asked to do: for example, the function returns a message with the reason for failure and/or launches a dialog to inform the Transporter using the application what went wrong. For further discussion on our definitions of success and failure, see the Developer Aiding documents.

System test plan

We defined system test procedures for the application. The implementation of the system tests using the Flutter testing framework is left as future work for the application and discussed in the Future Work section. These test procedures quickly become outdated as behaviour for the application changes.

We first identified three major experiences of the application:

1. Animal Transport Record creation, modification, deletion, and reading
2. Transporter account creation, modification, deletion, and reading
3. Application (system) features such as login, signup, internet access, etc.

Each of these three experiences is captured in the test procedures.

These tests were developed from a combination of our User Story/Journey Map stories, advice from the client and project mentors, and best software testing practices as described by Flutter (2021). The system tests are written in a generic step-by-step procedure

The written system tests were reviewed by the project mentor and approved accordingly.

GitHub actions workflow and use

We are able to run all of our tests automatically using GitHub Actions: the CI/CD tools offered to those using GitHub for public repositories. This is possible with the use of configuration files which define the workflow for running the tests. Our GitHub Actions files are under the .github/workflows directory in our repository. The workflow includes the checking out of the repository, starting runtime services, and then the Flutter test commands.

Per best team repository practice, no pull request or merging of code can be done unless all tests and associated workflow actions succeed. The workflow files force collaborators to maintain all tests, including system tests, all the time.

Our software testing results

All our Flutter tests (unit, integration) are passing (successful) as of April 1, 2021. The reader can directly confirm this information in our GitHub repository from the Actions tab or from the workflow status badges present on our README file.

Future Work

We recommend that the system test procedures be implemented once the behaviour of the application has been well-defined and existing code experiences less immediate changes; doing so would reduce developer frustration with changing tests drastically (the framework requires knowledge of the exact type of widget in some cases, which could change frequently) and increase confidence in the application's performance. The capability of interface testing with the Flutter framework is a powerful feature that should be used as soon as possible.

In addition, the integration of a testing tool that is capable of giving line, branch, and condition code coverage is recommended to test more thoroughly.

References

Flutter. (2021, April 1). *Integration testing.* https://flutter.dev/docs/testing/integration-tests

Flutter (2021, April 1). *Flutter test library.* https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html

Flutter (2021, April 1). *Testing Flutter apps.* https://flutter.dev/docs/testing

Flutter (2021, April 1) *Flutter testing cookbook.* https://flutter.dev/docs/testing

Pittet, S. (2021, April 1). *The different types of software testing*. Atlassian. https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing