Revised Test Plan - Receipt Scanner

Revisions

I had to completely ditch my preliminary test plan and do a new one to reflect a better understanding of testing. I have incorporated test suggestions from Stephanie’s review notes on a clearer definition of correctness and inclusion of a testing framework.

Intro

The unit testing for the Receipt Scanner component will attempt to exercise all non-trivial methods in the modules/classes *ReceiptFileCollector*, *ReceiptDataExtractor* and *ReceiptItem*. I plan on using Jest <https://github.com/facebook/jest> as my main testing tool for this component. Jest has great runtime speed and is most suitable for the Test Driven Development approach which I plan to implement with this component. In addition, I will also use Enzyme <https://enzymejs.github.io/enzyme/>, for whole component testing in conjunction and to supplement Jest.

I will create folders for every module and a single folder will contain the main class and its own test file in Jest. The unit test functions will run and provide test data when the test files are compiled and run. For more sophisticated testing with a multimodular interest, I will make use of Enzyme by creating a main test suite folder that will instantiate and make mocks of the different classes/modules to facilitate whole component testing.

Whole component functionality testing

The main functionality paths for this component that i’ll seek to exercise are:

1. If a user provides permission to access the camera: obtaining a grocery receipt photo directly from a camera capture, and using it’s URI, successfully extract data from it by calling the veryfi API with the URI as one of the parameters. The data we are interested in for every item is: *description, amount/quantity,* and the *units.*
2. If a user fails to provide permission for camera access a warning flag will be displayed and halt proceedings.
3. If a user provides permission to access the gallery: obtaining a grocery receipt photo file path directly from the gallery, and using it’s URI, successfully extract data from it by calling the veryfi API with the URI as one of the parameters. Then assert the expected data we are interested in for every receipt i.e *description, amount/quantity,* and the *units.*
4. If a user fails to provide permission for gallery access a warning flag will be displayed and halt proceedings.

Test cases for this would be:

1. Using 4 different short receipts from different stores with a few receiptItems, create a *ReceiptItemList* of all the receipt items with the properties *description, amount/quanity,* and *units* set correctly, then assert their values on the *ReceiptItemList* output from *ReciptDataExtractor.*
2. Using Enzyme, we’ll create a shallow version of *ReceiptFileCollector* and obtain a button response for denial of camera/ gallery access, and assert a warning/notification flag.

## Per-Class Unit Testing

ReceiptFileCollector

The functionalities to be exercised in this class are mainly the pickFromGallery() and pickFromCamera() routines. I’ll seek to exercise them by:

1. If user grants permission for gallery access, we should be able to get an image and have information on the uploadProperties variable, and a non null pointer on the receiptURI
2. If a user denies permission for gallery access, we’ll flag a warning message, and log the error on the console to stop any proceedings.
3. If a user grants permission for camera access, we would again expect no null pointers on the uploadProperties variable and the receiptURI
4. else, if not access granted we’ll flag an alert message

The test cases for this would be:

1. Using enzyme, we’ll create a shallow/simulate a version of the button response to grant or deny camera or gallery access, and for each case, assert an expected value, not null for uploadProperties and receiptURI when permission is granted for camera/gallery access and assert the alert message for when access is denied.

ReceiptDataExtractor

The most interesting method for this class is the creatReceiptItemList(). Nonetheless, testing this class will be progressive from testing ReceiptFileCollector, so that we’ll have as a pre-condition, the receipt URI that we'll use as a parameter for the method processReceipt.

The routines in this class can be exercised by:

1. If we have a valid receipt URI and a list of categories we expect that when the routine processReceipt() is executed the variable initialReceipt will not be a null pointer, and
2. The contents of initialReceipt will be used as a parameter for createReceiptItemList and that the output is a receiptItemList that has receiptItems with properties: description, quantity, amount, e.t.c originally contained in the json file as json objects.

The test cases for this class would include:

1. For createReceiptItemList: Using Enzyme and having a known/observed receipt receipt we’ll create a shallow instance of ReceiptFileCollector to obtain a receipt URI once all the tests for ReceiptFileCollector have passed. We’ll then create a list from observing the receipt and assert expected values for all the required properties on the list.
2. Using multiple different receipts, we’ll use enzyme once again to shallow instantiate ReceiptFileCollector each time, and assert values on our lists against the lists outputted by createReceiptItemList each time.

ReceiptItem

The methods in this class are quite trivial but we’ll use jester to assert that all the properties are set correctly and are accessed correctly by the get methods using our expected values.