PetInsights Test Plan

Presented by:
Pavni Bakhshi
Tosha Bapat
Mailys Terrier

Preface

PetInsights was tested using three methods. (1) Manual testing on project members' physical devices and emulators, (2) Automated testing with Xamarin.UITest, and (3) Random testing with the Exerciser Monkey tool. These combined methods offered both breadth and depth to the testing of our application.

Manual Testing

Upon adding new features to our application, we would individually conduct coordinated tests on our devices to get an initial idea of if the feature works and appears as expected on each different type of device (iOS, Google Pixel (Android), Samsung Galaxy S (Android).

Automated Testing

As the application was built on the Xamarin framework, we decided to take advantage of Xamarin's testing framework, Xamarin.UITest.

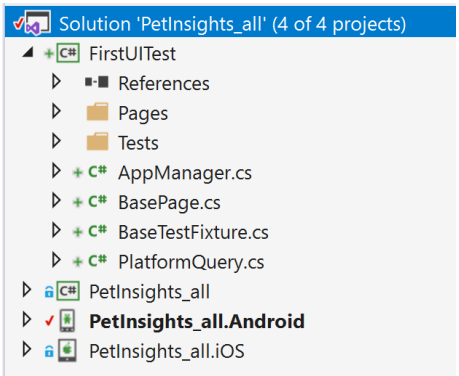
Background

Xamarin.UITest automates the user interface by activating controls on the screen and providing input anywhere a user would normally interact with the application. To enable tests that can *press a button* or *enter text in a box* the test code will need a way to identify the controls on the screen.

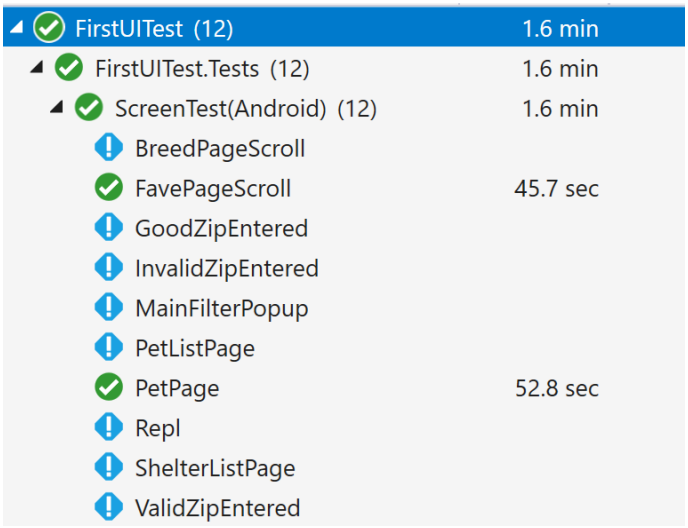
To enable the UITest code to reference controls, each control needs a unique identifier. We set this identifier by using the AutomationId property to all controls that are required for testing. For example:

```
<Button x:Name="b" AutomationId="MyButton" Text="Click me"/>
<Label x:Name="l" AutomationId="MyLabel" Text="Hello, Xamarin.Forms!" />
```

Xamarin.UITest gets added to the project folder alongside the Android and iOS folders as shown below:



Twelve tests were created to guarantee the navigation flow was as expected in each device, along with other UI testing scenarios.



Summary of the test cases, not all run in the picture

Sample Test Case for if tapping the filter button brings up the filter pop-up:

```
[Test]
0 references
public void MainFilterPopup()
{
    // Create a Xamarin.UITest that will search for our Label based on its AutomationId
    Query zipCodeLabelQuery = x => x.Marked("GetZipCode");
    Query FilterPopupQuery = x => x.Marked("SecondFilterLabel");


    var inputToValidate = "45140";
    app.EnterText("EnterLocation", inputToValidate);
    app.Repl();
    app.Tap("FindPetsButton");

    app.Tap("FilterButton");
    app.Tap("FilterButtonId");

    Assert.IsTrue(app.Query(FilterPopupQuery).Any());
}
```

Random Testing

Although automated testing is very beneficial, it risks a low code coverage as not all test cases or edge cases may be thought up by our own project members. Therefore the usage of a monkey automatic test-generator tool was explored with PetInsights.

Background

Android Monkey is a python-based testing script process that can be run by writing any specific python script or we can directly apply a test for the installed application using the command line.

Once the test process will be done then it will generate a log file of the events and crash if any. Generally, these testing tools are for the application developers while development in progress. A prerequisite for using this tool is to have Android SDK downloaded and installed on the system.

Monkey Exerciser will randomly check for user events such as clicks, touches, or gestures, as well as several system-level events.

The following are the commands that can be added to the usage of this tool to further specify what type of user interactions you would like to test on your application.

Command options reference

The table below lists all options you can include on the Monkey command line.

Category	Option	Description
General	`--help`	Prints a simple usage guide.
	`-v`	Each -v on the command line will increment the verbosity level. Level 0 (the default) provides little information beyond startup notification, test completion, and final results. Level 1 provides more details about the test as it runs, such as individual events being sent to your activities. Level 2 provides more detailed setup information such as activities selected or not selected for testing.
Events	`-s <seed>`	Seed value for pseudo-random number generator. If you re-run the Monkey with the same seed value, it will generate the same sequence of events.
	`--throttle <milliseconds>`	Inserts a fixed delay between events. You can use this option to slow down the Monkey. If not specified, there is no delay and the events are generated as rapidly as possible.
	`--pct-touch <percent>`	Adjust percentage of touch events. (Touch events are a down-up event in a single place on the screen.)
	`--pct-motion <percent>`	Adjust percentage of motion events. (Motion events consist of a down event somewhere on the screen, a series of pseudo-random movements, and an up event.)
	`--pct-trackball <percent>`	Adjust percentage of trackball events. (Trackball events consist of one or more random movements, sometimes followed by a click.)
	`--pct-nav <percent>`	Adjust percentage of "basic" navigation events. (Navigation events consist of up/down/left/right, as input from a directional input device.)
	`--pct-majornav <percent>`	Adjust percentage of "major" navigation events. (These are navigation events that will typically cause actions within your UI, such as the center button in a 5-way pad, the back key, or the menu key.)
	`--pct-syskeys <percent>`	Adjust percentage of "system" key events. (These are keys that are generally reserved for use by the system, such as Home, Back, Start Call, End Call, or Volume controls.)
	`--pct-appswitch <percent>`	Adjust percentage of activity launches. At random intervals, the Monkey will issue a startActivity() call, as a way of maximizing coverage of all activities within your package.
	`--pct-anyevent <percent>`	Adjust percentage of other types of events. This is a catch-all for all other types of events such as keypresses, other less-used buttons on the device, and so forth.
Constraints	`-p <allowed-package-name>`	If you specify one or more packages this way, the Monkey will *only* allow the system to visit activities within those packages. If your application requires access to activities in other packages (e.g. to select a contact) you'll need to specify those packages as well. If you don't specify any packages, the Monkey will allow the system to launch activities in all packages. To specify multiple packages, use the -p option multiple times — one -p option per package.

After running the Exerciser Monkey, the results can be found in a log as seen below:

```
#Intent;action=android.intent.action.MAIN;category=android.intent.category.LAUNCHER;launchFlags=0x10200000;
component=com.companyname.petinsights_all/crc64981426c216e4e19e.SplashActivity;end
    // Allowing start of Intent { act=android.intent.action.MAIN cat=[android.intent.category.LAUNCHER]
    cmp=com.companyname.petinsights_all/crc64981426c216e4e19e.SplashActivity } in package
    com.companyname.petinsights_all
:Sending Trackball (ACTION_MOVE): 0:(3.0,2.0)
:Sending Touch (ACTION_DOWN): 0:(789.0,472.0)
:Sending Touch (ACTION_UP): 0:(795.13525,477.0497)
:Sending Touch (ACTION_DOWN): 0:(674.0,2043.0)
:Sending Touch (ACTION_UP): 0:(686.0367,2043.3145)
:Sending Touch (ACTION_DOWN): 0:(922.0,1753.0)
:Sending Touch (ACTION_UP): 0:(927.9895,1754.7134)
:Sending Touch (ACTION_DOWN): 0:(1076.0,2072.0)
:Sending Touch (ACTION_UP): 0:(1077.166,2070.9062)
:Sending Touch (ACTION_DOWN): 0:(493.0,1261.0)
:Sending Touch (ACTION_UP): 0:(480.65967,1246.4891)
:Sending Touch (ACTION_DOWN): 0:(370.0,1538.0)
:Sending Touch (ACTION_UP): 0:(383.91525,1530.2421)
:Sending Trackball (ACTION_MOVE): 0:(1.0,-2.0)
    //[calendar_time:2021-04-19 16:54:43.085  system_uptime:311888955]
    // Sending event #100
```

Results

The Xamarin.UITest tool helped check for UI bugs in other devices, to guarantee that our UI remained consistent in all devices, screen sizes, and resolutions.

The Exerciser Monkey tool was great for investigating unexpected crashes, but tracing every test conducted by the tool would be a waste of time due to its difficult-to-read nature.