Test e2e Android

Appium et Espresso

Appium

test de blackbox

ce que vous voyez est ce que vous pouvez tester

Utilisé principalement pour des tests d'intégration de bout en bout et des flux d'utilisateurs entiers

peut être abstraite afin que le test écrit puisse être exécuté sur iOS et Android

Bien soutenu

prend en charge les tests parallèles sur plusieurs périphériques avec grille de sélénium

Expresso

test de boîte blanche / grise

peut modifier le fonctionnement interne de l'application et le préparer pour le test, par exemple enregistrer certaines données dans la base de données ou les préférences partagées avant de lancer le test

tester la fonctionnalité d'un écran et / ou d'un flux

Android uniquement

Bien soutenu

Pas de tests parallèles prêts à l'emploi, des plug-ins comme Spoon existent jusqu'à ce que le véritable support de Google apparaisse

Installation

- dependencies {

// Set this dependency so you can use Android JUnit Runner

androidTestCompile 'com.android.support.test:runner:0.5’

// Set this dependency to use JUnit 4 rules

androidTestCompile 'com.android.support.test:rules:0.5’

// Set this dependency to build and run Espresso tests

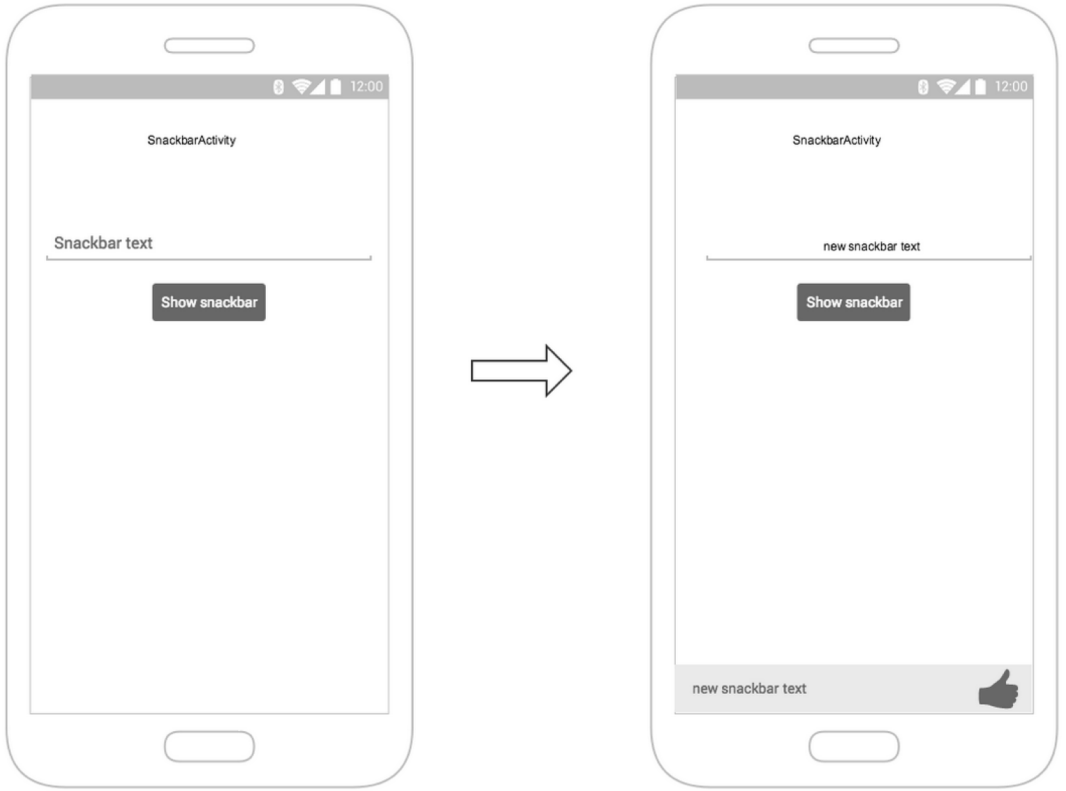
androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.2’

// Set this dependency to build and run UI Automator tests

androidTestCompile 'com.android.support.test.uiautomator:uiautomator-v18:2.2.2' }

Un premier test

L'écran contient:
champ de saisie de texte - **R.id.textEntry**
bouton qui montre le snack avec le texte tapé quand cliqué - **R.id.shownSnackbarBtn**
snackbar qui devrait contenir le texte saisi par l'utilisateur - **android.support.design.R.id.snackbar_text**



Un premier test

```
/**
* Testing of the snackbar activity.
**/
@RunWith(AndroidJUnit4.class)
@LargeTest
public class SnackbarActivityTest{
    //espresso rule which tells which activity to start
    @Rule
    public final ActivityTestRule<SnackbarActivity> mActivityRule =
        new ActivityTestRule<>(SnackbarActivity.class, true, false);


    @Override
    public void tearDown() throws Exception {
        super.tearDown();
        //just an example how tear down should cleanup after itself
        mDatabase.clear();
        mSharedPrefs.clear();
    }

    @Override
    public void setUp() throws Exception {
        super.setUp();
        //setting up your application, for example if you need to have a user in shared
        //preferences to stay logged in you can do that for all tests in your setup
        User mUser = new User();
        mUser.setToken("randomToken");
    }
```

Un premier test

```
    /**
    *Test methods should always start with "testXYZ" and it is a good idea to
    *name them after the intent what you want to test
    **/
    @Test
    public void testSnackbarIsShown() {
        //start our activity
        mActivityRule.launchActivity(null);
        //check is our text entry displayed and enter some text to it
        String textToType="new snackbar text";
        onView(withId(R.id.textEntry)).check(matches(isDisplayed()));
        onView(withId(R.id.textEntry)).perform(typeText(textToType));
        //click the button to show the snackbar
        onView(withId(R.id.shownSnackbarBtn)).perform(click());
        //assert that a view with snackbar_id with text which we typed and is displayed
        onView(allOf(withId(android.support.design.R.id.snackbar_text),
        withText(textToType))) .check(matches(isDisplayed()));
    }
}
```

onView (withXYZ) <- viewMatchers avec eux, vous pouvez trouver des éléments à l'écran

effectuer (click ()) <- viewActions, vous pouvez exécuter des actions sur des éléments trouvés précédemment

check (correspond à (isDisplayed ())) <- viewAssertions, vérifications à effectuer sur les écrans précédemment trouvés

Composants principaux

```
// withId(R.id.my_view) is a ViewMatcher
// click() is a ViewAction
// matches(isDisplayed()) is a ViewAssertion
onView(withId(R.id.my_view))
    .perform(click())
    .check(matches(isDisplayed()));
```

- **Espresso** – Entry point to interactions with views (via onView() and onData()). Also exposes APIs that are not necessarily tied to any view, such as pressBack().
- **ViewMatchers** – A collection of objects that implement the Matcher<? super View> interface. You can pass one or more of these to the onView() method to locate a view within the current view hierarchy.
- **ViewActions** – A collection of ViewAction objects that can be passed to the ViewInteraction.perform() method, such as click().
- **ViewAssertions** – A collection of ViewAssertion objects that can be passed the ViewInteraction.check() method. Most of the time, you will use the matches assertion, which uses a View matcher to assert the state of the currently selected view.
- Doc : https://developer.android.com/training/testing/espresso