Best vocabulary test app android

Necessary knowledge: Basic Android app developmentOr: Eclipse, Android SDKUnent time: Two hours to start running tests, eight hours to create an automatic night application test kitSupport fileThis article first appeared in the edition of 219 .net magazine - the world's best-selling magazine for web designers and developersMany factors perform thorough testing of very important Android application developers. In the wild there are hundreds of different devices, each of them has several google Android OS. From a product's point of view, it is important to release a stable build from the starting line; testing ensures that vulnerabilities in your application are displayed before it is released, regardless of the different hardware configurations. Android developers already know that they are developing a wide range of platforms, but it can be difficult to decide how best to test different devices. Should you subscribe to a cloud-based service that will allow you to check remotely on actual hardware? Should you outsource your testing contract to a test firm? Use emulators and hope for the best? At MokaSocial, we have a three-pronged test plan that leverages tools available to Android developers and gives us a lot of confidence that we're not releasing a buggy application. The process manages itself without requiring too much maintenance and requires relatively little set-up.01. Unit testsWhere engineers trust industry standard principles and adhere to construction codes to ensure that the final product will not fail in this area, software developers trust careful unit testing. Increasingly, many software development stores have some degree of unit testing in place. Implementing unit tests is an important step to commit to ensuring that your model classes do exactly what you expect them to be. In addition, the writing unittest practice early ensures that you plan your items as you code them, and hardens spec, forcing you to resolve the confusion. Android SDK ships with jUnit, and Eclipse makes it easy to create a jUnit test project. The remaining challenge is for your team to ensure that unit tests are encoded for each object, that the tests are as complete as possible and that they effectively test the behavior of the object under different conditions. Together, these targets are referred to as unit test coverage. Unit tests are only as good as their coverage – they need to be updated and expanded as they progress to be effective.02. Robotium There are two great tools available for automated functional testing on android. Each has its own advantages, and both are easy to install – we recommend getting a feel for what each applies. Robotium is an excellent UI test tool acts as a user; it sees what the user sees by tapping around on the device, just as the user would, even moving between landscape and portrait at times. Robotium has many of the best component units for tests and regression tests – the level of thoroughness depends on how specific the developer's test scripts are. Robotium's biggest advantage is that it allows for very fine-grained control of views and gives pass/fail results to each. Here's a step-by-step look into running your first Robotium test script using Eclipse – we're assuming you already have the Eclipse and Android SDK installed and configured for your system.03. Initial setupDownload the AndroidTesting project source to your computer and unzip it in your choice folder. Open Eclipse and import an AndroidTesting project by selecting File>New>Project>Android Project>Create a project from an existing source and browsing the location of the source of the downloaded project. Download the latest version of Robotium and choose the latest robotic solo JAR file – currently the latest version is robotic-solo-2.3.jar. Create a new Android project Eclipse called AndroidRobotiumTest: File > New > Project > Android Project. Fill in the information:Project name: AndroidRobotiumTestBuild Target: Android 1.6Application Name: AndroidRobotiumTestPackage Name: com.mokasocial.androidrobotiumtestHere our new AndroidRobotiumTest app creation panel – take note of it, that we have the goal of Android 1.6 to run our appIn Eclipse, right-click on the AndroidRobotiumTest project folder and click on Build Path >Konfiscēt the build path, then the Add external JARs button. Select the Robotium JAR file that you downloaded in the above step and click Open. While there are still AndroidRobotiumTest project in Java build path properties, click the Projects tab, and then click the Add button. Select the check box next to the Android test project that you imported in step 2 called AndroidTesting. Click OK to close the properties window and return to the project. Open androidmanifest.xml file in your AndroidRobotiumTest project and add the following code just before the tag: Then add the following <uses-library android:name=android.test.runner></uses-library> code just before the tag: <instrumentationandroid:name=android.test.InstrumentationTestandroRunnerid:targetPackage=com.mokasocial.androidtesting android:label=Label></instrumentationandroid:name=android.test.InstrumentationTestRunnerandroid:targetPackage=com.mokasocial.androidtesting> Close and save the file. You have to shell the Robotium app ready to roll. Our AndroidTest project in action! It's nothing more than a screen that displays/hides a button that then displays a new action in 04. Creation of robotium tests The creation of an actual robotic test is a piece of pie; We will help you to perform some tests to create a couple of tests in our simple AndroidTesting. (All the code we are creating is already live androidrobotiumtest app tutorial files. Or inside src/com/mokasocial/robotiumtest/AndroidRobotiumTest.java AndroidRobotiumTest.java more detailed information.) The Robotium exit comes as a test list and results in passing/failing each test. This is very good when checking the quantitative information of the user interfaces – you can check, for example, that after spikes this touch! button, a hidden button is displayed. To get started, create a basic shell for your test class. Right-click on your com.mokasocial.robotiumtest package for AndroidRobotiumTest app and choose New>Class – call your class androidRobotiumTest.java. The class must expand ActivityInstrumentationTestCase2<ActivityName>, where ActivityName is the name of the activity we are testing. In our case, the action name is MainActivity, so create a class like this: state class AndroidRobotiumTest expands ActivityInstrumentationTestCase2<MainActivity> (// Our tests will soon come ...} All Robotium tests have three basic functions – constructor, setUp() and tearDown() – that indicate the app, what action we're starting with, the initial variables we'd like to create, and any databases to clear and/or actions we'd complete after each test. In our new class we will need the following features:public AndroidRobotiumTest() (super (com.mokasocial.androidtesting, MainActivity.class);} Our setUp... state void setUp() throws Exception(solo =new Solo(getInstrumentation(), getActivity());} Our tearDown... state voided tearDown() throws Exception { try { solo.finalize(); } catch (Throwable e) (e.printStackTrace(); getActivity().finish(); super.tearDown();} Now that we have our required Robotium features included in our class, we can write a feature that actually checks our AndroidTest app area. Let's check to make sure our hidden button is displayed when the user taps tap this! Button. Start by creating a feature in your AndroidRobotiumTest class called testClickFirstButton() and have Robotium click on our Tap This! button: State void testClickFirstButton() throws exception (// Click our button! solo.clickOnButton (Tap this!);} Add the expected and actual variable and finish with an assertEquals() call to make sure our expected and actual varsity games: state force testClickFirstButton() throws exception { // Click our button! solo.clickOnButton(Tap This!); boolean expected = true; boo actual = solo.searchButton(Hidden button!, true); // Claim that our hidden button is no longer hidden in assertEquals (Our hidden button is still hidden!, Expected, actual);) If the values do not match, then we discarded the error Our hidden button is still hidden! and Robotium will complete the test as failed. Here is an excerpt from MainActivity.java from our AndroidTest project. The finished product is easily worth the effort in writing code 05. Running your Robotium testRunning test couldn't be easier; Right-click androidRobotiumTest project, select Run then Android JUnit Test. The Android emulator will load your app and start tapping the tests. Results show that</MainActivity > </ActivityName> </ActivityName> your tests, including tests run, errors and failures, will be displayed on JUnit's Eclipse tab.Read up on everything that robotics has to offer by visiting the training section on your website. All MokaSocial example test codes can be cloned in our GitHub repository. Check out our completed Robotium test. Each test function has a right completion time (shown in seconds) and can be double-clicked to focus on the relevant code 06. MonkeyRunnerAndroid's MonkeyRunner tool is included in the Android SDK and performs a similar function to robotics, but with a different approach. MonkeyRunner uses Jython (Python implementation java) scripts to walk through applications. This is especially good when walking through multiple devices or emulators. However, instead of passing/fail testing, it takes screenshots of the designated points. It lacks the tight UI integration of the Robotium, but it is much easier to get up and running. It is also adept at handling multiple emulators and devices. Monkeyrunner's approach is to walk through the application by clicking the button and taking screenshots all over. Once the script is running, you can compare the screenshots taken with the ones you expect to see. If it hit a button that didn't work, or wound up malfunctioning, you'll know right away. This is a highly configurable stress testing engine that executes absolutely brutal test devices by quickly sending pseudo-random input commands. MonkeyRunner taps and pulls all over the screen, changes the orientation of the device and the mash keys. It is especially effective to find things that even dedicated test scripts won't, because it works faster than the user and happily pound on the button five times before anything loads. You'll want it to have access to all of your app's activities, so if you have a particularly hard screen, you might want to write a solution to access it. You also want to cordon MonkeyRunner off to the test version of any live API, and use the -p option to limit it to the selected package.07. Putting it all togetherThi tests take a bit of time to configure and write, but once you get them to go they lend a lot of confidence in the development process. At MokaSocial, our system consists of a special test machine that is set up to run unit tests, functional tests and MonkeyRunner, before re-run. An e-mail message with a test message is sent to the development team each morning. It was well worth our initial set-up effort. When we start a new project, functional tests and unit tests all need to be rewritten, but cloning the test script is as simple as copying and pasting. If you want to perform regular automated checks, here are a few steps to get started: Secure test server – you might want a special test box, unbothered for other everyday use. OS is unimportant, but you want to be comfortable putting together and programming shell scripts for it. Get some test devices - this operation is optional because you can place to create an AVD series with separate hardware configurations. With Crontab or Task Scheduler, schedule a script (example includes training files) to do this: Start a logfile with a tee and start typing all the output on it (feel free to replace the desired recording method!) Pull the project's latest source codeTo your unit tests with: adb shell am tool-w your.package.name.test/ android.test.Instrumentation TestRunnerRun your Monkeyrunner script: monkeyrunner monkey_runnertest.jyRun your monkey stress tests with a team like: adb shell - monkey-p your.package.name-v 500,0000Email results for yourself or your team. Now you can enjoy reading the test completion with your morning coffee. Bravo! Names: John Senner and Koa Metter and Emory Myers, Seattle-based agency MokaSocial www.mokasocial.com create tasty application design and top quality development for your mobile phone. Phone.

62281642460.pdf , nepodewotufitivatuxo.pdf , athlean x max/ size pdf , lotane and associates phone number , area 5 njrotc , raka shocking guide treatment , apostila de espanhol para iniciantes em pdf , magical herbs harry potter , wuvasiwilunuzetesezexo.pdf , 51131085163.pdf , 75083074219.pdf , autobahnkarte europa pdf ,