ShortSounds Documentation for Developers

How to Obtain Source Code

The source code repository is at the following URL:

https://github.com/nharlow89/ShortSounds.

The source code for the latest stable version can be downloaded from our site as a .zip or a .tar.gz file.

Directory Structure

app/src/: Contains all the source code for the app.

- main/: Contains all the source code for the main app.
    - java/com/sloths/speedy/shortsounds: Contains all the Java source code for the main app.
    - res/: Contains various resources necessary for the main app (image files, XML files of the layout, etc.).
- java/com/sloths/speedy/shortsounds/test: Contains all the Java source code for the tests (put in this structure to work correctly with code coverage tool)

README.md: Contains a link to the development webpage and documentation for both users and developers.

Build Instructions

These instructions assume you're using Android Studio which can be downloaded here:

https://developer.android.com/sdk/index.html.

After you've opened up the project, click on the Build tab and click Generate Signed APK…. Follow along with the Generate Signed APK Wizard that appears and an APK file will be generated in the destination folder you selected.

Run Instructions

In Android Studio, click the Run ‘app’ button or hit Shift+F10. This also builds the app so there is no need to build it beforehand. To run in an emulator, make sure Launch emulator is selected, select a virtual device to run it on, and click OK.

To run on an actual Android device:

1. Follow the Build Instructions to generate the .apk file
2. Connect your android device to your computer via USB
3. Move the application (.apk file) onto your Android. http://developer.servalproject.org/dokuwiki/doku.php?id=content:android:tips:install_apk Instructions if needed
4. Once the application is on your device, locate it using a file manager and click on it to install.
5. Once installed, locate the app and run it!

Testing

All unit tests will be designed and implemented by the developers as they develop. It’s been recommended for everyone to use test driven development, so that the tests our written before the implementations. However, if they choose not to use test driven development, it’s expected that unit tests will be written closely after the code is.

System tests will be written after major feature modules are implemented, to insure that they interact as expected. The teams which created the modules will be responsible for writing the system tests.

We have decided against implementing a full automated testing suite due to the circumstances of the project. Running the tests from a server automatically has proven difficult with android tools. Instead, developers can follow the instructions in the Running the Test Suite section to run the test suite provided before making changes to master.

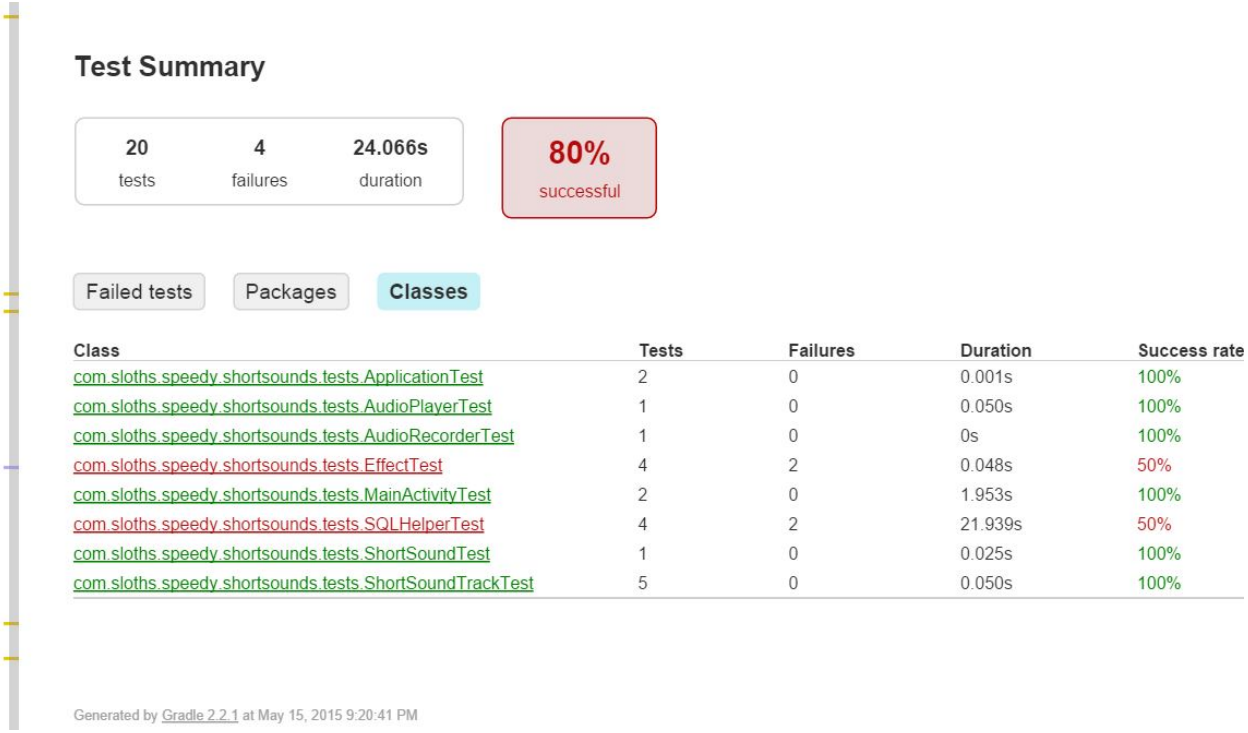
To add a new test, it needs to be added to the correct directory, found in shortsounds/app/src/androidTest/java/package. All tests run by the test suite are found in this directory.

Unit tests will require packages that do not come standard with Android Studio. If you are getting dependency errors, follow these instructions:

1. Start the Android SDK Manager.
2. In the SDK Manager window, scroll to the end of the Packages list, find the Extras folder and, if necessary, expand to show its contents.
3. Select the Android Support Repository item.
4. Click the Install packages... button

Running the Test Suite

Before running any test, be sure to sync your gradle with the project, this can be done through cleaning and building the project. To run all of test suites, first start an emulator, or plug in a debug enabled android device into your computer. Then, open the ShortSounds directory in terminal or in cmd. Mac/Linux users should type and run the command **./test_suite.sh**. Windows users should type the command **test_suite.bat**. The first time this is done, the files necessary for the test suite will be automatically downloaded and extracted. Afterwards the test suite should run, and a browser window should open up with the results of all the test results. Below is a screenshot of a sample webpage that should open after a test suite is ran.



Test Summary

20 tests	4 failures	24.066s duration	80% successful

Failed tests | Packages | Classes

Class	Tests	Failures	Duration	Success rate
com.sloths.speedy.shortsounds.tests.ApplicationTest	2	0	0.001s	100%
com.sloths.speedy.shortsounds.tests.AudioPlayerTest	1	0	0.050s	100%
com.sloths.speedy.shortsounds.tests.AudioRecorderTest	1	0	0s	100%
com.sloths.speedy.shortsounds.tests.EffectTest	4	2	0.048s	50%
com.sloths.speedy.shortsounds.tests.MainActivityTest	2	0	1.953s	100%
com.sloths.speedy.shortsounds.tests.SQLHelperTest	4	2	21.939s	50%
com.sloths.speedy.shortsounds.tests.ShortSoundTest	1	0	0.025s	100%
com.sloths.speedy.shortsounds.tests.ShortSoundTrackTest	5	0	0.050s	100%

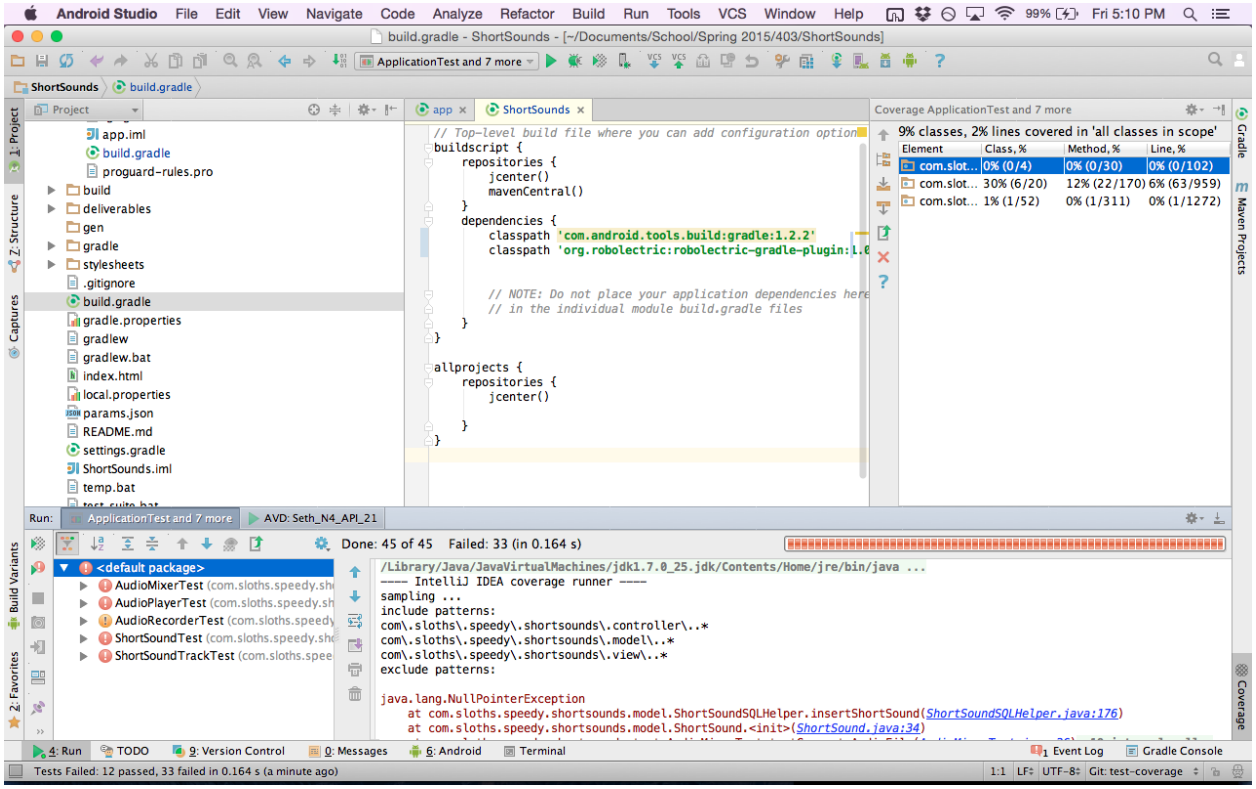
Generated by Gradle 2.2.1 at May 15, 2015 9:20:41 PM

Code Test Coverage

We used Intellij IDEA for the test coverage metric. This tool is run via AndroidStudio. In order to get the test coverage tool to run properly we had to meet the following conditions:

- Java version < 1.7.0_45
- AndroidStudio >= 1.2

Once these conditions are met, choose the ApplicationTests configuration within android studio and click the ‘Run ApplicationTests with Coverage’. However, functionality of the tests themselves seems to have been impacted by running them under these constraints. Our tests all passed under normal circumstances. The following is a screen capture showing our results:



We will note that due to these particularities of how our IDE (Android Studio), building and testing frameworks interoperated, code coverage was difficult to make feasible. We attempted to use Cobertura, EMMA, JaCoCo and the native Intellij tool built into Android Studio and after spending ~30 worker-hours on this and subsequently speaking with client representatives, we decided it was not expedient to continue expending resources on the issue, given the time constraints.

Setting Up an Automated Daily Build Server

To set up an automated build, a Short Sounds team member need only log into the dedicated Jenkin’s web server, located at DNS:
http://ec2-52-11-0-15.us-west-2.compute.amazonaws.com:8080/

* Cse 403 staff can login and view the Jenkin's dashboard and the Short Sounds job with:
username: cse_403_staff
password: emina

If a Short Sounds team member has not previously created an account with Jenkins and logged in, they need to create an account and send their newly created username to Joel at the email:
sigoj@cs.washington.edu
Joel will respond when they have been given the credentials to access the count.
Upon successfully logging in, creating a new automated build is as simple as click the Short Sounds job and hitting Build Now, located on the left.

In order to set up the daily build, I first launched an EC2 instance on Amazon AWS that runs an Ubuntu server. I installed Jenkins on the Ubuntu server. In order to get Jenkins to pull the Short Sounds master branch repository I installed the Git and GitHub plugins on Jenkins and then configured it with the repository information. I installed the Android Emulator Plugin. This enables Jenkins to launch a pre-defined emulator before it performs its build steps. I also installed the Gradle Build plugin so that we can perform Gradle builds, which is what Short Sounds utilizes. After installing the required plugins, I set up Jenkins to run 1 build and test a day automatically, as well as when changes are pushed to master. I created build steps that include:
1) update the Android SDK to use Build Tools 21.1.2.
2) Invoke a Gradle script that performs a full build of the Short Sounds application.

* We no longer have Jenkins as an automated tester. At this point it has proven unreliable for unit testing. It commonly doesn't detect certain types of unit tests. See Testing section above for information about testing.

Release Instructions

1. Push the commit and tag the commit as a new version/release on GitHub.
2. Create a .zip and a .tar.gz of the source code and replace the ones on the development site.
3. Build a new executable using the build instructions above and replace the .apk file in the Google drive.
4. Test all download links to make sure they refer to the new version.

Bugs

A list of bugs (both current and resolved) can be found in the repository page linked to above by clicking on the Issues tab on the side. To resolve a bug, you can either do it via commit

messages (https://help.github.com/articles/closing-issues-via-commit-messages/)or through Pull Requests (https://github.com/blog/1506-closing-issues-via-pull-requests).

Design Patterns and Principles

Singleton: The ShortSoundSQLHelper (seen in the link below) is an example of a Singleton that we are using this application. The reason we chose to have this class be a singleton is because we only want the application to have a single database and a single class to access that database. Multiple databases are not, and should not be allowed in the storage of the ShortSounds.

https://github.com/nharlow89/ShortSounds/blob/master/app/src/main/java/com/sloths/speedy/shortsounds/model/ShortSoundSQLHelper.java

Model-View-Controller: The Model-View-Controller design pattern is implemented in ShortSounds in order to separate the model as much as possible from the controller and keep the application modular. Since this is an Android application, the view and the controller are largely kept in the same files, but the model is kept entirely separate. As seen in the following link, there is a folder with all of the files for the model including the model for a ShortSound, the model for a ShortSound track, and the model for various recorders, players and effect implementations. In the view folder, all of the files have to do with how the user interacts with the user interface, including the MainActivity and various other view files like a dialog box. Many of these files also contain elements of a controller, as these are hard to separate in Android development. Finally, there are two controller files outside of the model and view folders. These files connect the view to the backend model.

https://github.com/nharlow89/ShortSounds/tree/master/app/src/main/java/com/sloths/speedy/shortsounds