Installing and Configuring Android Studio

Further instructions will assume you have installed and configured the latest version of Android Studio. If you have not yet installed it, you can download it here:

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

When installing, select the custom install option and install all optional components. If at any point you are missing components required for project sync and/or build, Android Studio will give you a notification prompting you to install them. If it does, click on the notification and install them, then try again.

When importing the Finders Keepers project into Android Studio, make sure you import the Finders Keepers folder from the repository, not the whole repository. The repository contains a docs folder which has development documentation that is not part of the application.

Obtaining the Source Code

You can find the Finders Keepers repository on Github at the following link:

https://github.com/dowelc/finders-keepers

Simply click the “clone or download” button and click “Download Zip” to download a Zip archive containing the source code for the latest commit. To download the source for a different commit, click the “commits” button, followed by the “<>” icon next to the commit you want to download. Then, click “clone or download” and “Download Zip” to get the source.

Directory Structure

The “docs” folder at the root of the repository contains various developer documents, including UML class and sequence diagrams, weekly reports, UI mockups, etc. The FindersKeepers folder contains the source for the Finders Keepers Android App. Inside it, you will see various gradle files and a gradle folder which contain configuration files for building the application. The “AndroidBootstrap” folder contains the source for Android Bootstrap library components which we are using in our project.

The “app” folder contains the source for our actual application. Within it, the “androidTest” folder contains instrumentation tests for our app, which will execute on the testing device or emulator. The “test” folder contains unit tests which will execute on the development machine.

The “main” folder within “app” contains two folders: “java”, which contains all the java files which control our application, and “res”, which contains xml files which store the layouts for all the

screens and widgets in our application and various global constant variables. “res” also contains various image resources used in the application.

Back at the root of our repository, we also have the “scripts” folder, which contains various scripts for deploying our backend.

How to Build Finders-Keepers

Finders-Keepers has a relatively simple build process, but has several prerequisites for operation:

On the backend:
- Python 2.7
- Flask for python (Run ‘pip install Flask’ with a valid Python 2.7 installation)
- MySQL

On the frontend:
- Java 8 SDK
- Gradle
- Android Studio (Recommended -- Comes with integrated Gradle + Android SDK)

For the backend, after setting up the database server, all that is needed is to install Flask and run the main server script: “python server.py,” which will create a local instance of the API server on port 5000. If Android Studio is not installed, the project can be built by running Gradle from inside the FindersKeepers folder (run ‘gradlew’ from a shell or command prompt). This will create an apk in the /app/build/outputs/apk folder. Alternatively, you can install Android Studio and import the FindersKeepers folder as a project, then build from inside Android Studio. We recommend this option to avoid potential issues with Java or Gradle configuration, and to make deployment to device or emulator easier.

How to Test Finders-Keepers

Finders-Keepers uses the Gradle build system on the client side to automate testing. To build the project and run all unit tests automatically, run ‘gradlew test’ from a shell or command prompt in the FindersKeepers folder (Note: Requires that Gradle is installed). If you are using Android Studio, you can instead run test suites individually or in bulk from within the program (right click the test(s) or test folder -> run).

How to Release a New Version

New versions should be released whenever a major feature is implemented or a significant amount of bugs have been fixed. When appropriate, the following steps should be taken:

- BEFORE versioning, a complete build and test should be performed, and the app loaded both on an emulator and on a live device. If any of the tests fail or other issues are encountered, fix them before proceeding any further.
- After a successful test, update the versionName and versionCode constants in the build.gradle app file. The versionCode should be incremented for each version, and the versionName should be set to something appropriate with a higher version number and, if not a final release, a short code indicating the release status (0.5b for a beta for instance, or 0.9rc for a release client.)
- Next the project should be built with the new version code, and committed to the repository including any resulting apks.
- On Github, create a with the version name previously chosen. A copy of the apk should be included separately in the release. The download link on the website will always automatically point to the latest release on github -- if this is not desired (for a test release, for instance), mark the release as a 'pre-release' when publishing.

Bug Management:

We use Github issues as a bug + feature tracker. Our issues page is located here: https://github.com/dowelc/finders-keepers/issues .

Design Patterns Used in Finders Keepers:

One design pattern our application uses is the singleton pattern. This is used by the UserDataHolder object in app/src/main/java/cse403.finderskeepers/data/UserDataHolder.java. There is only one instance of this class in our application. It has a private static field which holds this instance, and a private constructor which is only used once within itself. This means only one instance of the class can ever exist, making it a singleton. To use the instance, users call UserDataHolder.getInstance(). The class is used to hold information on the current user, including location, avatar, and id. This information is used in almost every API call the app makes, meaning it is important that many classes have access to it. Putting it in a static singleton, rather than passing it around, makes it easily accessible throughout the application.

Our overall design architecture, from frontend to backend, is also representative of the Model View Controller pattern, a type of behavioral pattern. In a Model View Controller architecture, the view is what is presented to the user and what the user interacts with the system through. The controller processes user input and provides data from the model to the view in a way which can be presented to the user, and the model holds all data involved in the application. In our application, the GUI of the Android app is the view. It is what the user interacts with the application through. The controller is both the back end of the Android app, including UI listeners and API handlers, as well as the API server. The API server does a fair bit of computation for things such as searches, which is what makes it part of the controller rather than the model. our MySQL server is the model, since it is where all of our data is stored. You can view our Software Design Specification on our team website (linked from our Github page)

to see how each of these components interact together to form the MVC architecture as described here.