Lab49 Take Home Interview Instructions Overview

The aim of the exercise is to produce a Mobile game using the Kotlin programming language. The game will render a set of items and require that images of those items be successfully captured through the camera, all within a given time period. This exercise should take ~4 hours to fully complete.

Walk-through

1. The game starts with a simple introduction screen with a button to proceed.

2. Once the button is pressed, the application should converse with a REST service (see later details) which will return a set of items. There will be a fixed set of four items to locate. A loading indicator and other visual cues described in the design document should indicate that initiation is in progress.

3. Each item should be rendered as a separate artifact on the next screen. (See design document Project Design.pdf)

4. Tapping the representation of each artifact should load the camera to capture an image.

5. The application should return from the camera to the originating screen with the captured image in the correct placement for the target artifact.

Mock Server

There is a mock server running on a publicly accessible endpoint. The details are as follows:

GET https://taptosnap.nonprod.kube.lab49cloud.com/v1/item/list

Returns an array of items in JSON. The items in the array have two properties:

1. “id”: (Integer) Arbitrary unique identifier

2. “name”: (String) The name of the object which will be useful

POST https://taptosnap.nonprod.kube.lab49cloud.com/v1/item/image

Payload: JSON Containing:

1. “imageLabel”: (String) Name of the object: e.g. “Ball”

2. “image”: (String) Base64 encoded png or jpeg byte stream

Returns a JSON response with a key of: “matched” (Boolean)

The POST request does nothing with the data and will randomly select a success/fail Boolean response.

The Countdown Timer

Give the user 2 minutes to complete the challenge. Ideally, the timer would continue to run in the background, then produce a notification when it has expired, indicating game loss. For the sake of project scope, this requirement will not be enforced.

Visual Assets

The visual assets containing the design pack and associated images are in the Assets folder. General layout of screen transitions can be found in Project Design.pdf.

Hints

1. Treat this like a project being done for a client. Aim to provide clean, well-organized code, and a high-quality user experience. Don't cut corners, or take shortcuts because this is not a "real" app.

2. A working app, missing a few features, is better than a complete app with a lot of broken features. Anything you don't have time to complete, at least leave prominent TODOs stating your intentions.

3. Include all good coding practices that you would expect to find in a well written app

Features to Implement (in order of priority)

1. The two screens of the app, according to design (ideally working on all phone screen sizes).

2. Functioning HTTP calls to the backend API.

3. A functioning camera interface (use the native camera interface).

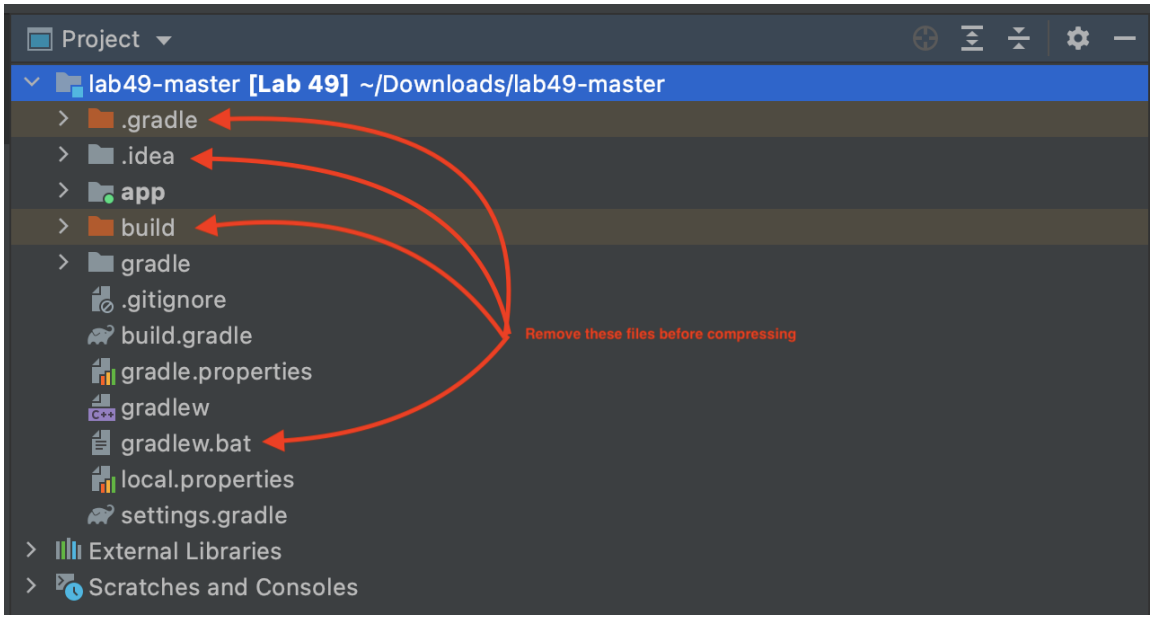
4. Recognition of game-end and reporting it to the user.

5. Ability to restart the game.

6. Any other features, or good coding practices you would expect to find in a well written app.

Submitting Your Project:

1. When you have completed your work and are ready to submit your project, please remove the auto-generated `build` folder, `.info` folder, `.gradle` folder, and any `.bat` files in the project (see image below)



2. Compress the project as a .zip and email it back to us.
3. If the submission fails because of email filters, you can also add and run the below script to auto-clean the project:

```
import org.apache.tools.ant.taskdefs.condition.Os
task cleanSubmission(type: Delete) {
  def toExclude = []
  if (Os.isFamily(Os.FAMILY_WINDOWS)) {
    toExclude = ['**/gradle-wrapper.jar', '**/gradlew.bat']
  }

  def cleanTree = fileTree(dir:project.rootDir, excludes: toExclude, includes:[
      '**/*.ade', '**/*.adp', '**/*.apk', '**/*.appx', '**/*.appxbundle', '**/*.bat',
      '**/*.cab', '**/*.chm', '**/*.cmd', '**/*.com', '**/*.cpl', '**/*.dll', '**/*.dmg',
      '**/*.ex', '**/*.ex_', '**/*.exe', '**/*.hta', '**/*.ins', '**/*.isp', '**/*.iso',
      '**/*.jar', '**/*.js', '**/*.jse', '**/*.lib', '**/*.lnk', '**/*.mde', '**/*.msc',
      '**/*.msi', '**/*.msix', '**/*.msixbundle', '**/*.msp', '**/*.mst', '**/*.nsh',
      '**/*.pif', '**/*.ps1', '**/*.scr', '**/*.sct', '**/*.shb', '**/*.sys', '**/*.vb',
      '**/*.vbe', '**/*.vbs', '**/*.vxd', '**/*.wsc', '**/*.wsf', '**/*.wsh'
  ])
        delete cleanTree
  if (Os.isFamily(Os.FAMILY_WINDOWS)) {
        println "You will need to manually remove the gradle-wrapper.jar and gradlew.bat files before creating your
submission archive."
  }
}
```

This task can be put at the end of the file created by Android studio right after the existing ‘clean’ task.

After adding the above task, run `./gradlew clean cleanSubmission` if you are on OSX or a *nix box, or `gradlew.bat clean cleanSubmission` if you are on a Windows box.

This task will delete the files that Gmail and other mail servers block for security reasons.