NTNU

Norwegian University of
Science and Technology

IMT3673

Project report

Group 2 - Colour Vision Defieciency Test

Tom Roar Furunes(223686)
tomrf@stud.ntnu.no

Manuel Jesús Bravo García(470335)
mjgarcia@stud.ntnu.no

Tomasz Rudowski(471181)
tomaszmr@stud.ntnu.no

Sjur Sutterud Sagen(140695)
sjurssa@stud.ntnu.no

May 3, 2018

1 Introduction

This project was proposed by the Colour Lab at NTNU Gjøvik, with the supervisor being Philip Green. The final project specifications were decided on after two meetings between the team and the supervisor.

A Colour Vision Deficiency application can help to detect colour vision deficiency. The main goal is that it can run standard tests, like basic Ishihara test, with some variance in plates, like e.g. randomised sequence of plates. Those test are usually done using printed plates in a controlled test environment. The basic variant of the Ishihara test we are implementing consists of the plates with numbers only.

2 Requirements

This section presents an overview for requirements after elicitation process during meetings with the project supervisor. For the evaluation purposes requirements were marked with a current status when this report is submitted.

The main goal of the project is an app that can run standard vision deficiency tests, like a basic Ishihara test. An Ishihara test can have some variance in plates pictures, randomised sequence of plates etc. The app should use the plates with numbers and input as free text not buttons that may suggest an answer as in other apps. Potentially it could support wider range of colors than sRGB (then bound to minimum Android 8.0, API 26+), but this will not be implemented in this version. Following tasks have been identified; some are marked as optional that represents a lower priority or alternative solution to consider and if necessary explained later in details:

- (DONE) Create a picture set that represents one test (pack it along with .json files into a zip file) and place it on public available URL, consider different sets of pictures for randomising tests.
- (NOT DOING - insufficient time to prepare a set of vector graphics that could have a randomize colours when running a test) (optional) create one set of .svg files and randomize colours on local device.
- (DONE - json file on public URL) REST API service that responds with a JSON (array of TestInfo objects) - (optional) mock, firebase, mLab+GO. (chosen

alternative solution: list of tests as a json file on given URL that represents possible API response)

- (DONE - downloading .json file) Fetch Test list from that API or .json file.
- (DONE) Create Activity/Fragment with a list of tests available to download (populate list when test list fetched, on element click start download service)
- (DONE) Local database (stores Test Info for tests downloaded and available to run)
- (DONE) Download Service (create folder named based on test ID, unzip pictures and .json files there, run Local Database Service to add TestInfo to local tests when downloading complete)
- (NOT DOING - used internal storage to local tests, i.e. no initial test data on app installation) (optional) Create folder named with pictures and .json files in Assets. Use the for GUI/Run Test Activity as a demo/default. Replace with internal storage later (or keep both sources).
- (DONE) Create Activity/Fragment with a list of downloaded (local) tests (run Local Database Service to populate list, on element click start run test activity)
- (NOT DOING - insufficient time, high-end devices binding) (optional) add mobile screen colour calibration (e.g. Enhancing Graphics with Wide Color Content)
- (DONE - shown on test Welcome Screen) (optional) implement Activity to adjust screen brightness before test.
- (DONE) Create an Activity to run a local/downloaded test (need to be implemented as a package for each test type to ensure that questions and input from users fit the test e.g. integer input for Ishihara test, sort rectangles that have colour variations for another test etc). All necessary data (pictures from local storage) will be get after "Run test" button clicked (don't fetch resources from internal storage in advance).
- (DONE) There is a limit of three seconds to give an answer on the analogue version of the Ishihara test. After discussion we decided that this should be changed to a higher value because of keyboard/input from a test subject, instead of a voice answer.

- (DONE) (optional) Show the same plate multiply times, use e.g. a Plate priority or randomise (implementation proposal Plate object in Test.plates more than once, this could be decided by test author, who may use multiply Plate objects in plates.json file).
- (DONE) Create JSON parsers that could create objects from .json files
- (DONE) PlateFragments that are designed for a concrete test type (consider timer for an answer on Ishihara test).
- (DONE) Create logic to summarise test results based on documented Ishihara test interpretation.
- (DONE - no receiver of POST Request, payload created) Create payload of ResultSet and upload/POST as json to results server (requested to not store in local storage/database - security).
- (NOT DOING - agreed not to implement this) (optional) Use as a part of metadata reading from sensors etc.(light environment). Discussed with supervisor: tests metadata not that important, may be added later if found useful.

3 External Services

The application is supposed to run remotely prepared tests. To obtain it we assumed that some external services need to be implemented and integrated with.

3.1 List of Available Tests

The first of them should provide an API that responses with an array of JSON objects. Those objects will be used to populate the list of test available for downloading. Test metadata will be stored on local storage (database) when test is downloaded. Listing 1 presents data structure that is an expected response.

Listing 1: Expected JSON response for a list of available tests - array of TestInfo objects

```
[ {TestInfo}, ... ]

TestInfo
{
  "id":               String,   // uuid()
  "name":             String,   // test name
  "description":      String,   // test description
  "resources":        String,   // URL to <TestID>.zip file
  "resultsServer":    String,   // URL to POST results to
  "type":             String,   // test type eg "Ishihara"
  "version":          int,      // if used
  "firstPlate"        int       // null if not relevant
}
```

3.2 Test Resources

The second service is a storage for .zip files (URL coresponds with TestInfo.resources attribute), that consists of plate pictures and additional .json files (test type specific). We are expecting a test would uploaded by researcher to resource server packed in a .zip file. TestInfo.id could be generated by the same server that is used to upload set of pictures; so all pictures would be packed together with additional metadata.

The zip file consists of:

- a set of plate pictures - filenames corespond with Plate.filename in the data structure.
- plates.json - an additional JSON file with an array of Plate objects. If it is requested to show any of plates more then once, the solution would be to have multiply Plate objects in the array in the file - picture files are not duplicated. As in IshiharaPlate example, for basic red-green deficiency attributes "protanStrong" and "deutanStrong" should be the same. If "extra' attribute is set to "true" it means that plate is an additional one to differentiate between Protan and Deutan (currently not in use - explained later in this report).

Listing 2: The plates.json file consists of an array of Plate objects

```
[ {Plate}, ... ]

IshiharaPlate
{
 "id":            int,    //  plate id
 "filename"       String  //  file path <TestID>/"filename"
 "normal":        int,    //  expected input value for normal vision
 "protanStrong": int,    //  expected input value for vision deficiency
 "deutanStrong": int,    //  expected input value for vision deficiency
 "total":         int,    //  expected input value for total color blindness
 "extra":         bool    //  if true this plate may be in the calculation of additional results
}
```

- thresholds.json - an additional JSON file with extra data concerning results interpretation. In an Ishihara test example number of plates with an answer indicating normal vision is considered; two values are used in the JSON file: normal - number of answers above this limit means ”normal vision”, deficiency - the number of answers below it means ”red-green” colour deficiency. If number of given answers is between those two values - the result is uncertain. We use 5 seconds limit for an answer, but it could be overidden by a timeLimit attribute. In an analogue test of 15 plates (numeric plates only with no repetitions)[1], those values are 13 and 9, but since the number of plates could be various it is needed to provide thresholds for results calculations.

Listing 3: IshiharaThreshold object in thresholds.json

```
{
"normal":        int,    //  expected input value for normal vision
"deficiency":    int,    //  expected input value for vision deficiency
"timeLimit":     int     //  optional, to define "show plate" time (other then default)
}
```

Currently both TestList.json and an example .zip file are available on a public URL and are downloaded to an internal storage on a mobile when chosen. The services (to be develop in the future) would be hosted on Colour Lab servers; therefor we decided to not create any unnecessary dependencies in the code (e.g Firebase); this choice also allow a researcher to prepare all the files needed to create and publish a new test in a simple text editor.

[1]http://www.dfisica.ubi.pt/ hgil/p.v.2/Ishihara/Ishihara.24.Plate.TEST.Book.pdf

3.3 Results Submission

The third service would be an REST API that could be used to gather results from tests (current implementation would make a HTTP POST Request with a JSON payload based on test results)

4 Implementation

4.1 Data and Package Structures

We decided to divide project files as following (fig. 1):

- Core package includes all common and base classes (including main activity fragments, download services and database handling).
- Ishihara package consists of all classes that are specific for this test type; among them visualisation (test specific input/output), results interpretation. All other test types should be implemented in this way i.e. as a separate package.

Class attributes that corresponds with JSON structures shown previously are also presented. Test includes a list of Plates that are available to download from URL indicated by resources attribute. Each type of test needs both Plate and Result classes extended in order to properly interpret values marked as awaited and given by a test subject; in the Ishihara test example on a picture they are numbers that represents normal vision and deficiencies (red-green deficiencies with care for protan/deutan results and total colour-blindness). To ensure that all (also future types of tests) will be shown on the mobile screen properly, we need to implement a class TestActivity to match test type.

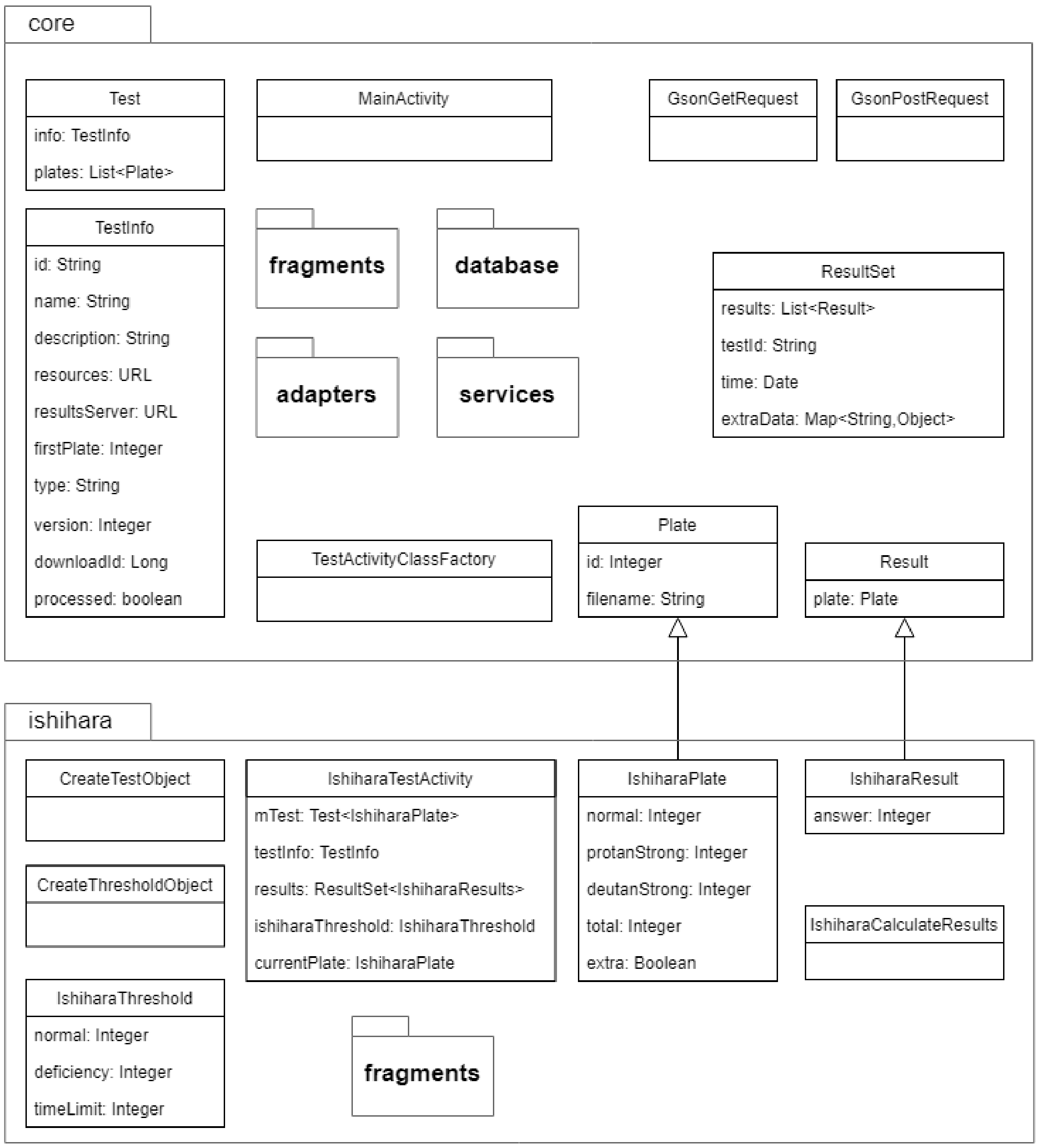

Figure 1: Package and data structures

If an extra answer is needed for a specific plate, it might be placed in Result class extension; this will also allow to create a test type where answers could be more complex (more steps) or of different type (String, Array etc). Summary of results may consider thresholds.json file, on a picture below an example of a class (IshiharaThreshold) that the JSON file is marshalled to; two attributes are used to give a general answer (normal, uncertain, deficiency); counters for Deutan and Protan answers can additionally indicate strong Deutan/Protan deficiency if possible based on given answers - if not, assume a general red-green deficiency. ResultSet in addition to a set of Results may contain metadata (not defined yet) that could be e.g. local time, light intensity while test taken or general personal data (gender, age etc) if accepted by user and needed for research. If chosen to publish, the ResultSet will be send to URL defined in TestInfo.resultServer (this need to be implemented separately if results are to be gathered).

4.2 Fetching Test Data

This section explains the complete process of fetching data into a local device.

The TestList.json is fetched using Volley[2].

When we have the TestList.json, which contains URL to the .zip, this get downloaded by DownloadTestService, then processed by ProcessDownloadService and the database is updated.

4.2.1 Download Services

This uses the Android DownloadManager[3] to download the zip to the downloads directory in your external storage(you cannot use DownloadManager to download to internal).

4.2.2 Internal Storage

We then use the ProcessDownloadService to unzip the files to to the internal storage. For unzipping we use ZipInputStream[4].

[2]https://github.com/google/volley

[3]https://developer.android.com/reference/android/app/DownloadManager

[4]https://docs.oracle.com/javase/7/docs/api/java/util/zip/ZipInputStream.html

4.2.3 JSON Parsing

JSON parsing is done with google-gson[5], to marshall JSON objects (String) to Java objects.

4.2.4 Database

All classes concerning database handling are placed in “database” package. We have used Room database that stores TestInfo objects (the class is annotated as @Entity). The application contacts a singleton instance of the database using TestInfoDAO interface. Specific tasks are defined in the package as extensions of AsyncTask.

4.3 UI

Genaral proposal of UI that was accepted during meeting is presented in the project Wiki[6]. The main consider was to avoid colours that could distract user taking the test and don’t implement additional question to distinct strong and mild (for both Protan and Deutan).

4.3.1 Main Activity

The main activity has the necessary logic in order to change between the different fragments. This is done through several methods that will create those fragments.

4.3.2 Adapters

We have one adapter (TestListAdapter) that is added to the RecyclerView[7] for both the list of local tests and, and downloadable tests.

[5]https://github.com/google/gson
[6]https://github.com/tomme87/Colour-Vision-Deficiency-Test/blob/master/docs/GUI.md
[7]https://developer.android.com/reference/android/support/v7/widget/RecyclerView

4.4 Running an Ishihara Test

Here we are following the entire process of running a test from both user (UI) and app (data flow) perspective.

4.4.1 Initialise Test Data

We need to fetch data from internal storage in order to create objects necessary to run a test. This is realised by following AsyncTasks:

- CreateTestObject - adds plates.json file marshalled to a set of IshiharaPlates.
- CreateThresholdObject - run after the first one and marshals thresholds.json file

When tasks have completed jobs the first plate could be shown.

4.4.2 IshiharaTestActivity

This activity gives the application support to run Ishihara tests. This activity will make use of two fragments: one that shows the plates of a test while the other shows the results of the test. The activity has several methods that will choose a random plate to be shown, collect the answers of the user and set any constraints that are necessary to run the test. The application can be run by people with colour vision deficiency, so it uses white, black and grey colours to not distract them.

4.4.3 Limitations

To allow adjustment of the time limit for an answer, we added an attribute timeLimit (Integer) to a thresholds.json file; if this attribute is null or negative time limit is set to a default value. Additionally we are using only Ishihara plates with numbers, so the user input is numeric.

4.4.4 Result Calculation

The algorithm we presented during meetings (more details in project wiki) with supervisor was considered interesting, but we were requested to implement a standard interpretation (as available in Ishihara book[8]). We use counters for each parameter (normal vision, deficiency-protan etc.) and compare with available IshiharaThreshold values to generate summary message to the test subject.

4.4.5 Result Summary

The final fragment shown that displays the appropriate result message. The user can start the result submission and close the test, or simply close the test.

4.4.6 Result Submission

If the user chooses to do so, the result could be sent to a researcher. This is done by marshalling the ResultSet to JSON and posted to a URL found in TestList.json. This requires that the researcher have a server which can receive this json etc. we have not implemented this server.

5 Conclusions

We are planning on future work with project after its submission deadline, due to necessary transition of code base to Colour Lab. Additionally the project has been documented in Wiki[9] section of the repository, the section was used for both communication channel to present solution proposals (that might extend some of the sections from this report) and to keep track of requirements and progress status.

[8]http://www.dfisica.ubi.pt/ hgil/p.v.2/Ishihara/Ishihara.24.Plate.TEST.Book.pdf
[9]https://github.com/tomme87/Colour-Vision-Deficiency-Test/blob/master/docs/Home.md