Report for Mobile Programming (CISC3002, University of Macau) Course Project: “King of Karaoke” – a quiz app composed of various features

KE SIYUN alias Garfield
(DB927871)
garfield.ke@connect.um.edu.mo

Abstract- A project implementing an Android application for a music quiz game. The game update problem sets including question texts, segments of songs, and cover images from online locations, and use those recourses downloaded to local storage to generate questions. The app allows users to log in and updates the highest scores achieved by users to online ranking.

***Keywords-* OKHttp3, JSON, AsyncTask, Authentication, ViewPager, Fragment, Music player**

Introduction

This course project aims to implement a tiny quiz game application on Android, which allows the players to choose the correct title or singer of a song, based on a given segment of the song and an image of the album cover. This app downloads a problem set prepared by the author from an online source. The author summed up experiences from previous assignments/ labs and designed a special way to perform the process (details provided in the report).

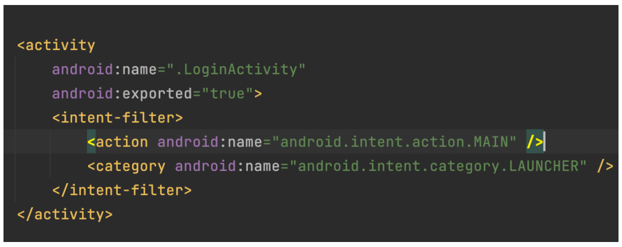
This application also enables the users to leave their name by logging in with their email, or google account (through Auth0), when the players submit their answer, the application compares their score with the current highest score it grabbed from an online resource (JSONBin). Besides, this project’s implementation also involves many other android development features like the intent handling while passing data, lifecycle

Design and Implementation

This app is not a big work, but it consists of various features and the author applied some complex designs for algorithms to fully meet the requirements. These factors will be explained in detail in the following sections.

Log-in/log-out handling

The players log-in to the app through the API provided by Auth0. They can achieve this with an email address or single-sign-in with a google account – it’s especially convenient while testing the application through an AVD emulator logged in with the google account. The main purpose of log-in for this application is to record the identity of the player.



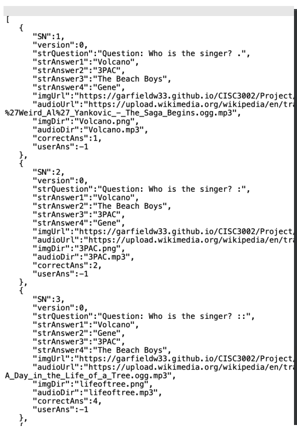
```
<activity
    android:name=".LoginActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

The log-in is handled in the LoginActivity, the manifest file was modified thus making it the entry upon starting up the application (instead of Main). The authentication credential is processed by the API of Auth0. This activity then handles the updating of the problem set (described in a later part). After all is prepared, it uses an intent to start the MainActivity.

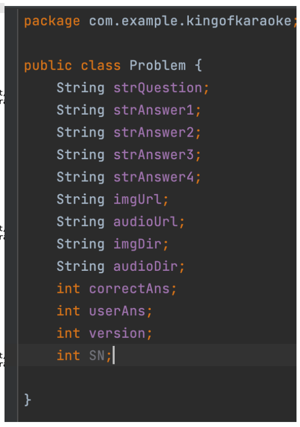
Problem-set updating

The problem set consists of 20 songs/singers. It will be downloaded to the internal storage of the application in its initial start-up. And it checks an online location every time a user logs in, to see if there is any update to the problem set.

To have a better understanding, we shall first notice that I placed the album cover images, and segments of the song, in a GitHub repository (actually a GitHub Page), together with a magical .json file. This JSON file serves as a map for the online and local resources.



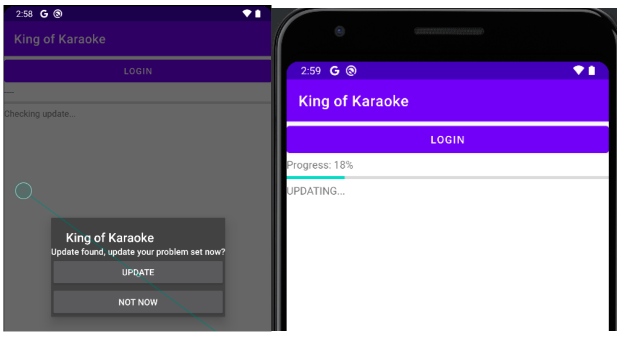
```
[
   {
      "SN":1,
      "version":0,
      "strQuestion":"Question: Who is the singer? .",
      "strAnswer1":"Volcano",
      "strAnswer2":"3PAC",
      "strAnswer3":"The Beach Boys",
      "strAnswer4":"Gene",
      "imgUrl":"https://garfieldw33.github.io/CISC3002/Project
      "audioUrl":"https://upload.wikimedia.org/wikipedia/en/tr
%27Weird_Al%27_Yankovic_-_The_Saga_Begins.ogg.mp3",
      "imgDir":"Volcano.png",
      "audioDir":"Volcano.mp3",
      "correctAns":1,
      "userAns":-1
   },
   {
      "SN":2,
      "version":0,
      "strQuestion":"Question: Who is the singer? :",
      "strAnswer1":"Volcano",
      "strAnswer2":"The Beach Boys",
      "strAnswer3":"3PAC",
      "strAnswer4":"Gene",
      "imgUrl":"https://garfieldw33.github.io/CISC3002/Project
      "audioUrl":"https://upload.wikimedia.org/wikipedia/en/tr
      "imgDir":"3PAC.png",
      "audioDir":"3PAC.mp3",
      "correctAns":2,
      "userAns":-1
   },
   {
      "SN":3,
      "version":0,
      "strQuestion":"Question: Who is the singer? ::",
      "strAnswer1":"Volcano",
      "strAnswer2":"Gene",
      "strAnswer3":"3PAC",
      "strAnswer4":"The Beach Boys",
      "imgUrl":"https://garfieldw33.github.io/CISC3002/Project
      "audioUrl":"https://upload.wikimedia.org/wikipedia/en/tr
A_Day_in_the_Life_of_a_Tree.ogg.mp3",
      "imgDir":"lifeoftree.png",
      "audioDir":"lifeoftree.mp3",
      "correctAns":4,
      "userAns":-1
   },
```

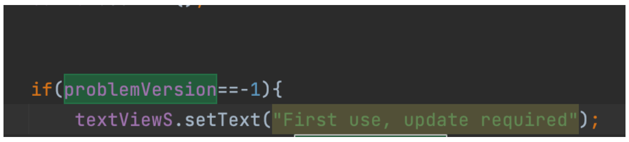


```
package com.example.kingofkaraoke;

public class Problem {
    String strQuestion;
    String strAnswer1;
    String strAnswer2;
    String strAnswer3;
    String strAnswer4;
    String imgUrl;
    String audioUrl;
    String imgDir;
    String audioDir;
    int correctAns;
    int userAns;
    int version;
    int SN;

}
```

The structure of the JSON file is shown in the image above. It has a few greatest usages, for instance, the update method relies on the integer "version", to determine whether there's an update for the problem set. In fact, it downloads the JSON file directly after log-in and compares the new JSON's version code with the old version code in the local.

It asks the user to decide whether to perform an update in case a higher version code appeared, if so, it downloads the images and audios from the location marked in "imgUrl" and "audioUrl", with OKHttp. And renames the files with the name "imgDir" and "audioDir", these filenames are also referenced by the MainActivity to access the downloaded files hence a copy of the JSON is also kept locally. However, notice that if the application does not find a JSON file in an internal location, it will consider it as an initial start-up, and will directly perform the download without asking the user in the pop-up.



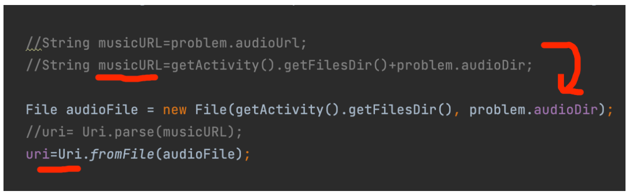


```
if(problemVersion==-1){
    textViewS.setText("First use, update required");
```

This JSON is not stored in the JsonBIN, since the author wishes to easily update it (release new problem sets) without changing its URL.

Music playing

The author simply planted the music player we implemented in the previous lab class for this project. The music player is placed in the fragment of question (accessible in ViewPager), and it receives the content to play from the adapter as other components do. Besides, another major difference from the previous player is that the player refers to a local file instead of an online resource. The author read some documents[7] and managed to transfer the file route into a URI for the player to access the audio and it shows up with a good result.

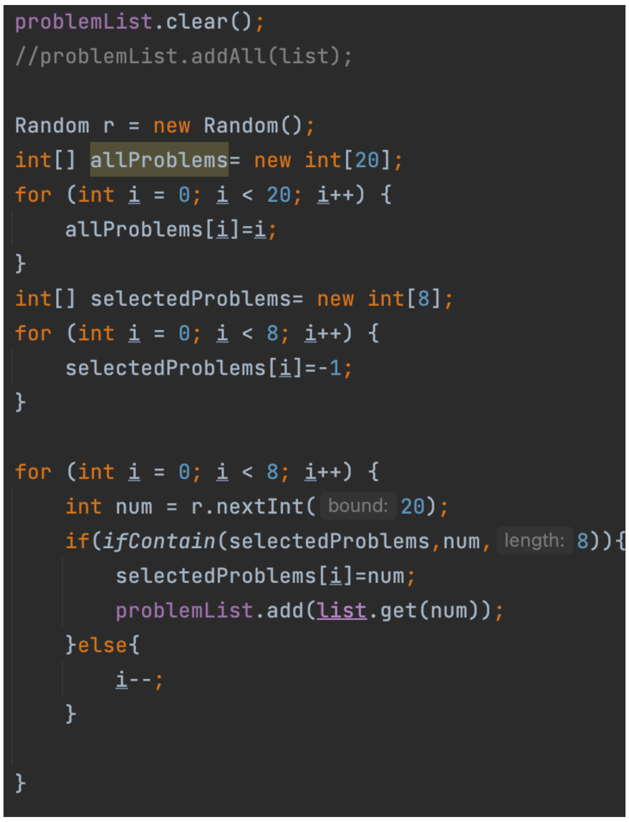


```
//String musicURL=problem.audioUrl;
//String musicURL=getActivity().getFilesDir()+problem.audioDir;

File audioFile = new File(getActivity().getFilesDir(), problem.audioDir);
//uri= Uri.parse(musicURL);
uri=Uri.fromFile(audioFile);
```

Question drafting

A total of 20 questions are included in the problem set described in the JSON file. We need only 8 of them in one quiz.

The author implements simple drafting using the Random class of Java. The function randomly picks the indexes from 0 to 19 and records the selected indexes, it generates another one once there's a duplicate until it has 8 random indexes for questions among 20. Also thanks to the help of Gson, which read the JSON as a list of question data, we can easily pick the question according to its index.



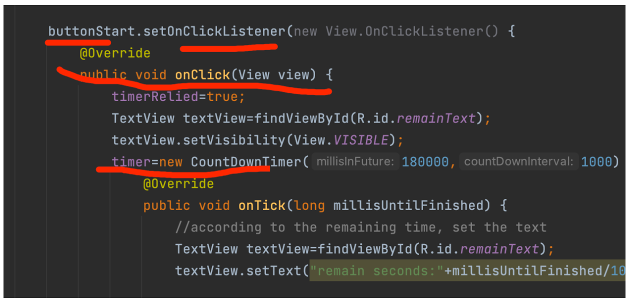
```
problemList.clear();
//problemList.addAll(list);

Random r = new Random();
int[] allProblems= new int[20];
for (int i = 0; i < 20; i++) {
    allProblems[i]=i;
}
int[] selectedProblems= new int[8];
for (int i = 0; i < 8; i++) {
    selectedProblems[i]=-1;
}

for (int i = 0; i < 8; i++) {
    int num = r.nextInt( bound: 20);
    if(ifContain(selectedProblems,num, length: 8)){
        selectedProblems[i]=num;
        problemList.add(list.get(num));
    }else{
        i--;
    }

}
```

Besides, the drafting is performed every time the player clicks "Start Game" in the main activity, hence the player will not have the same questions after a submission.

Counter down timer

This counter-down timer is inspired by the work in AsyncTaskTest. It uses the CountDownTimer to generate a counter after the "Start Game" button is tapped. The author put the methods to end a game and submit the result into the onFinish() override or the timer, thus the quiz will be automatically submitted for calculation score once the timer stopped.

However, we need to notice a point that if the user manually submitted a game by clicking the submit button, we shall end the timer immediately or it will affect the result display or the next round of the game. Thus I changed the original function form of calling the timer into an object after the onClick function of the "Start Game" button, hence I can easily stop the timing by timer.cancel().

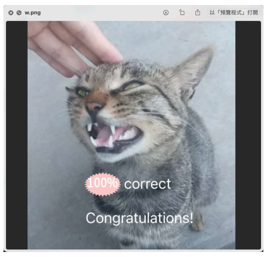


```
buttonStart.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        timerRelied=true;
        TextView textView=findViewById(R.id.remainText);
        textView.setVisibility(View.VISIBLE);
        timer=new CountDownTimer( millisInFuture: 180000, countDownInterval: 1000)
            @Override
            public void onTick(long millisUntilFinished) {
                //according to the remaining time, set the text
                TextView textView=findViewById(R.id.remainText);
                textView.setText("remain seconds:"+millisUntilFinished/10
```

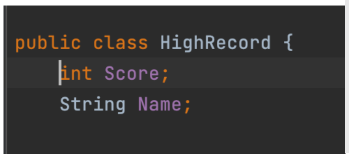
Apart from the quiz timing, I also used the timer to handle some results from async tasks thus needing some waiting.

Score and upload the best score.

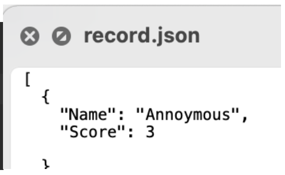
Once the user clicked the submit button or the time is used up thus submitting automatically, a method (submitScore) will be called and it will start a new score activity, where the users' answers (choices in radio groups) will be examined with the correct answer marked in the problem list. It generates a summary and if the player reached 8/8 correct, an additional greeting image will show up.



The best score is also kept in a JSON file. This JSON file is not stored on GitHub Page but JSONBin for easier updating.

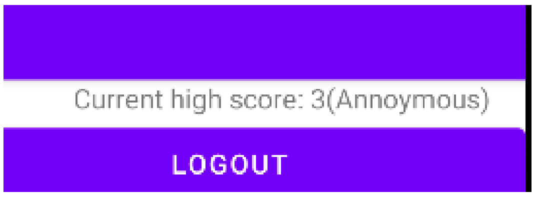


```
public class HighRecord {
    int Score;
    String Name;
```

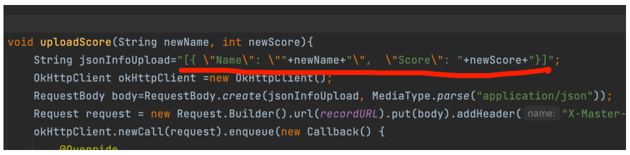


record.json

```
[
  {
    "Name": "Annoymous",
    "Score": 3
  }
```



The author designed such a way to process: Download and process the JSON containing the highest score and the name each time when starting/ resuming the main activity (with OKHttp and Gson), and this info is displayed in textViews in the Main activity and score activity. And if the scoring activity determined the player achieves a better score, it will use a master key of the JSONBin, generate a request and cover the original record.



```
void uploadScore(String newName, int newScore){
    String jsonInfoUpload="[{ \"Name\": \""+newName+"\",  \"Score\": "+newScore+"}]";
    OkHttpClient okHttpClient =new OkHttpClient();
    RequestBody body=RequestBody.create(jsonInfoUpload, MediaType.parse("application/json"));
    Request request = new Request.Builder().url(recordURL).put(body).addHeader( name: "X-Master-
    okHttpClient.newCall(request).enqueue(new Callback() {
        @Override
```

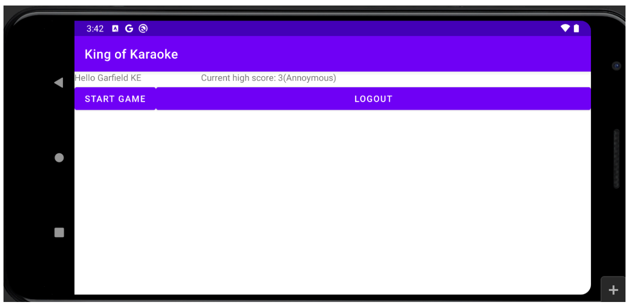
This system can only contain one highest record currently. But I suppose adapting it for multiple records and even additional information like time, if required, is still not difficult.

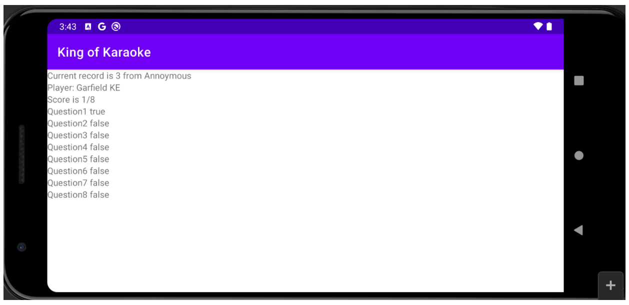
Meanwhile, there's a notable change in the Main Activity and Score Activity. In previous assignments, players may re-attempt the same questions after submission and exit from the scoring activity. Current applications introduce onPostResume, a "Start Button" button, with other algorithms making components invisible or disabled, to meet the requirement to refresh activity every time.

The ViewModel and rotation

This application has its data stored in some common lists. Its view model class contains a problem list. (and a String for "level", not very useful in this project but left from previous assignments, I kept this for possible level selection in the future.) The problem list kept all the data read from Json stored in local internal storage. Other data which is not used much is not in the model

Referring to problems led by rotating the phone, I used the linear layout with proper wrapping size. Thus does not have many rotation issues. One possible improvement is that scrolling is needed to view some longer pages.





```
Current record is 3 from Annoymous
Player: Garfield KE
Score is 1/8
Question1 true
Question2 false
Question3 false
Question4 false
Question5 false
Question6 false
Question7 false
Question8 false
```

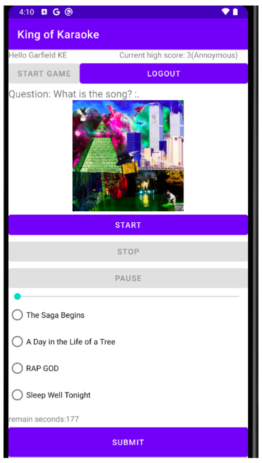
Exception Handling

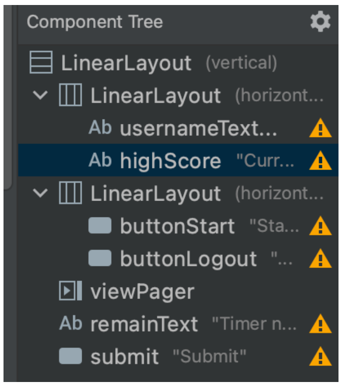
Many exceptions took place in the login part of my project since it needs to download much data from various resources. Most exceptions themselves reveal information like local file-not-found suggests an initial update (download) is needed. For the exceptions users may encounter, like turning off(leave) the app while playing music, the onStop/ onDestroy methods will deal with that and keep the data integrity.

But the Internet might be a problem I suppose. Almost every part of the program needs accessing the internet for information. Actually, besides the initial start-up, offline use may be acceptable in other cases, if disabled the features like online ranking. But my current application doesn't support offline operation for the entire process.

User Interface

The interface is placed mainly under LinearLayout. Although very simple to arrange, I managed to place them as clearly as possible.





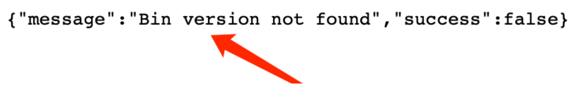
The author also managed to achieve a more smoothing process of using the app. i.e. placing useful buttons in sequence, and disabling/enabling components to enhance visual attraction.

Discussion

Besides the given requirements in the project description, I would suggest several possible improvements for the application. -The level of a game may be selected by the user and different level results in a different number of question (currently 8 questions). The author left some variables and methods in the functions of reading JSON problem set and random drafting, so as to modify into an application with variable level selection.

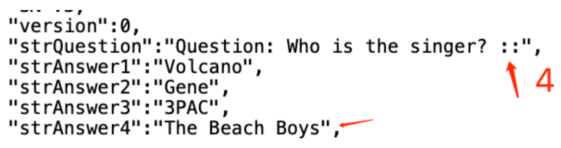
Besides, the current application can store only one highest record (and the name of the player), which is enough for an experimental project, while more records and other additional info concerning the records like the time to complete and the date of attempts may be considered for real application.

There's a problem left in the project, that the updating of JSONBin results in the change of the referencing URL. While considering how to seek a change of version for updating, the author considered treating this as a feature, i.e. to try to access the updated URL (adding "/1", "/2 "… after the bin URL) to see if there's an update. But it wasn't implemented and as you may see, I implemented a version number directly recorded in problem set JSON with a non-change URL.



```
{"message":"Bin version not found","success":false}
```

There is also a tip for testing the program, the correct answer to the question is marked with dots (. / : / .: / ::) beside the question mark. I designed this for a more convenient test process.



```
"version":0,
"strQuestion":"Question: Who is the singer? ::",
"strAnswer1":"Volcano",
"strAnswer2":"Gene",
"strAnswer3":"3PAC",
"strAnswer4":"The Beach Boys",
```

Acknowledgments

The audio and image files used in this image are selected from a Wikipedia site [1], some of them are not in the public domain but the author uses these resources solely for academic purposes. This project utilized an authentication service from Auth0 for non-commercial development.

Some design of the author's work was inspired by some online articles/guides, the websites are listed in the reference.

References

[1] Wikipedia non-free audio samples
https://en.wikipedia.org/wiki/Category:Wikipe
Wik_non-free_audio_samples

[2] 使用 Glide 加載图片系列之一从不同的数据源加载图片
https://www.imzhiqiang.com/2015/10/27/Load-images-from-difference-data-sources/

[3] How to stop CountDownTimer in android
https://stackoverflow.com/questions/40203827/how-to-stop-countdowntimer-in-android

[4] android 利用回調函数在对話框中傳遞数据, CC 4.0 BY-SA ,
https://blog.csdn.net/u013564742/article/details/49914815

[5] Android studio Dialog 彈出式对話框, CC 4.0 BY-SA,
https://blog.csdn.net/haolangtaiye/article/details/80309724

[6] Access app-specific files
https://developer.android.com/training/data-storage/app-specific#java

[7] FileProvider
https://developer.android.com/reference/androidx/core/content/FileProvider