BooYah! - The moment-based media rating application

Group 5 final Project for Dalhousie University's Fall 2018 edition of CSCI 4176/5708: Mobile Computing.

Instructor: Dr. Tami Meredith

Members

- Jigar Joshi (B00812722)
- Shivani Desai (B00799169)
- Bola Okesanjo (B00876268)
- Vivek Shah (B00799155)
- Ryan Stevens (B00695460)

GitLab Repository

The GitLab repository is here (https://git.cs.dal.ca/rstevens/booyah). To request access, contact Ryan Stevens at ry366081@dal.ca (mailto:ry366081@dal.ca)

Important: In order to use *BooYah!*, the server must be running. Please contact Bola Okesanjo at o.okesanjo@dal.ca (mailto:o.okesanjo@dal.ca) or (902-448-0790) to arrange a time for it to be running.

Project Summary

BooYah! is a time-based media rating application for Android that allows users to create quantitative feedback about particular moments of movies, tv shows, music, and other time-based media. Users rate particularly impactful moments bad ("Boo!") or good ("Yah!"). Ratings for a BooYah! session are assembled into graphs that show the parts that the user liked and disliked. When multiple people rate a piece of media, those ratings are aggregated to show the total number of likes and dislikes for moments of the media.

Audience, Purpose, and Benefits

Popular media rating platforms like Rotten Tomatoes (https://www.rottentomatoes.com) provide quantitative (numeric) data about audience's overall impressions about media, but provide only qualitative (written) feedback about **what** makes the media so great or so dismal. Content viewers must then manually parse the feedback from other users to determine if it has elements they will like. Similarly, content creators must carefully read that information to learn how to create better content in the future. *BooYah!* solves this problem by allowing viewers to quickly see the moments that were liked and disliked by the audience.

Users of *BooYah!* will do so with any combination of these three goals in mind:

- **Rate Media**

  Typical users, mainly teens and young adults, will use *BooYah!* as a way to express their opinions about media they consume. They will sit down in their homes with the media, and the application open beside them. When they find themselves liking or disliking a particular moment, they will express that emotion by pressing "Boo" or "Yah". After the session, they will likely view the graph they have generated and reflect on how good (or bad) the media they just consumed was. If the user wants to share their rating graph on social media, they will take a screen-shot of their device and share that photo.

- **See opinions to determine if they want to consume the media**

  Other primary users, mainly teens, young adults, and tech-savy adults, will use BooYah! to decide whether or not they wish to consume a particular piece of media. They will search for the media and view the aggregate graph of ratings about that media. Based on information they gleam from the graph, they will be able to make an informed decision about whether or not to proceed with consuming the media.

- **See opinions so they can produce better media**

  Content creators who seek to improve their quality of content will use *BooYah!* to learn what their audience liked and disliked about existing media, and tailor their new content to include more moments like the ones rated good, and fewer moments like the ones rated bad.

Libraries

The following libraries are used in this project:

Android libraries

The following are the Android libraries used to create the mobile application:

- **square-retrofit:** Retrofit is an HTTP client for Android and Java. We chose it over *volley* because, unlike *volley*, it can send POST multi-part form requests which we needed for automatic media-time recognition. Source here (https://square.github.io/retrofit/)

- **google-gson:** Gson is a Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Source here (https://github.com/google/gson)

- **retrofit-gsonconverter:** Gson converter is an Android library used by Retrofit to serialized Java objects to JSON format by using Gson as the serialization layer. Source here (https://github.com/square/retrofit/tree/master/retrofit-converters/gson)

- **material-edittext:** Material EditText is an Android library that contains fancy widgets for Android Edit Text fields. Source here (https://github.com/rengwuxian/MaterialEditText)

- **android-graphview:** GraphView is an Android library that is used to display diagrams and plot graphs. We used it to plot the likes and dislikes for a given media. Source here (http://www.android-graphview.org/)

Icon library

- **font-awesome:** Font awesome is a popular website that provides free icon sets for websites. The icons are licensed under the Creative Commons Attribution 4.0 International license. We use these icons for the buttons on the application. Source here (https://fontawesome.com/icons?d=gallery)

Python API libraries

The following are the Python packages used to create the REST API for the project:

- **flask:** Flask is a web development micro-framework for Python. It is highly extensible using individual libraries or other 3rd-party libraries to perform tasks like database ORM, serialization etc. Source here (http://flask.pocoo.org/)
- **flask-sqlalchemy:** Flask-SQLAlchemy is a Python library that integrates Flask with SQLAlchemy. SQLAlchemy is a popular Python ORM for SQL. Source here (http://flask-sqlalchemy.pocoo.org/2.3/)
- **flask-marshmallow:** Flask-Marshmallow is a Python library that integrates Flask with Marshmallow. Marshmallow is an ORM-agnostic serialization library for Python objects. Source here (https://marshmallow.readthedocs.io/en/3.0/#)
- **flask-jwtextended:** Flask-JWTExtented is a Python library that enables a Flask application to use JSON Web Tokens (JWT tokens). JWT is a JSON based standard for creating HTTP access tokens. Source here (https://flask-jwt-extended.readthedocs.io/en/latest/)
- **dejavu:** DejaVu is a Python library that performs audio fingerprinting and automatic audio recognition using a database of cryptographic fingerprints. Source here (https://github.com/worldveil/dejavu)
- **pydub:** PyDub is a simple Python library that is used to manipulate audio files. It does this using ffmpeg. Source here (https://github.com/jiaaro/pydub)
- **pytube:** PyTube is a lightweight Python library for downloading YouTube videos. Source here (https://python-pytube.readthedocs.io/en/latest/)
- **celery:** Celery is a robust asynchronous task queue for Python. It uses a distributed messaging queue such as Redis or RabbitMQ to pass messages between a set of tasks and an application. We use it to run resource-intensive API requests asynchronously. Source here (http://www.celeryproject.org/)
- **rabbitmq:** RabbitMQ is a popular distributed message broker that we use to pass messages between the API and long running tasks. It does this using the Advanced Message Queuing Protocol (AMQP). Source here (https://www.rabbitmq.com/)

Installation Notes

None. However please contact Bola Okesanjo at o.okesanjo@dal.ca (mailto:o.okesanjo@dal.ca) or (902-448-0790) in order to set up the API for the app.

Code Examples

The following are some problems that we encountered when creating the Android application

1. **Displaying aggregate likes and dislikes for a piece of media**

We needed to figure out how to merge the individual user ratings for a media so that our graph would show an aggregate rating (like or dislike) for every second the media had been rated.

To solve this problem, we assigned a numeric value of 1 to each second that a user liked, and a value of -1 for each second that the user disliked; if users A & B like media X at the 10th second, then the 10th second has a value of 2. If another user C also dislikes X at 10secs, then that second has a numeric value of 1. This lends itself to easy interpretation on a graph. If an overwhelming number of users like a second of the media, the bar for that second is above 0, and vice versa.

To implement this, we used a `Hashmap` to store each second (key) and its numeric value (value). After getting the list of likes & dislikes - for a media - from the API, we loop over both lists and add each second that has been rated to the map. In the process, we also increment/decrement the value of each second that exists in the map with a particular user's like/dislike. The code shown below illustrates this.

```
HashMap<Integer, Integer> map = new HashMap<>();

//for the same key increase the value by 1 -- Likes
for (int userIdx = 0; userIdx < aggregateLikes.size(); userIdx++) {
    List<Integer> seconds = aggregateLikes.get(userIdx).seconds;

    for (int secIdx = 0; secIdx < seconds.size(); secIdx++) {
        Integer key = seconds.get(secIdx);
        if (map.containsKey(key)) {
            Integer mapValue = map.get(key);
            map.replace(key, mapValue + 1);
        } else {
            map.put(key, 1);
        }
    }
}
```

2. **Determining the type of JSON response received from an audio recognition request to the API**

The API returns either a `Status` or a `Recognition` JSON object whenever we request the status of an audio recognition task. It returns a `Status` object whenever it is still trying to recognize a media or it returns the result of the audio recognition whenever it is done. JSON examples are shown below.

```
{
    "uuid": "c4c5b11f-b959-4faf-b8a1-dc2e4da19264"
}
```

a. Status object

```
{
    "duration": 133,
    "id": 3,
    "match_time": 0.3682839870452881,
    "name": "Dilbar",
    "offset": 65.89823
}
```

b. Recognition object

In Java we cannot declare both object as the return value of a `Retrofit` call to the API. To solve this problem we declared the return value of the corresponding call as a generic Java Object. In order to determine which of the 2 object an API response is, we take the string representation of the object and check it for any of the fields shown above. If it has a `uuid` field, then it is a `Status` object and vice versa. If it is a `Recognition` object, we deserialize it using JSON and pass it to the appropriate activity. The following code snippets illustrate the solution.

```
StatusResponse status = call0.execute().body(); // synchronous call since
we're on different thread
 Call<Object> call1 =
apiService.getRecognizeMediaStatus(userToken.getToken(), status.getUuid());
 Object result = call1.execute().body();
 int retry = 3;
 while (retry > 0 && result.toString().contains("uuid")){
     Thread.sleep(200);
     call1 = apiService.getRecognizeMediaStatus(userToken.getToken(),
status.getUuid());
     result = call1.execute().body();
     retry--;
 }

 if (result.toString().contains("name")){
     final RecognitionResponse recogres =
RecognitionResponse.createFromObject(result);
 }


public static RecognitionResponse createFromObject(Object obj) throws
MalformedJsonException {
     String[] arr = obj.toString().split(",");
     for (int i =0 ; i < arr.length; i++){
         if(arr[i].contains("name=")){
             String[] split = arr[i].split("=");
             arr[i] = split[0] + "='" + split[1] + "'";
         }
     }
     String json = String.join(",", Arrays.asList(arr));
     JsonReader reader = new JsonReader(new StringReader(json));
     return new Gson().fromJson(reader, RecognitionResponse.class);
 }
```

Problem 3: Numeric overflow with nanoseconds

BooYah!'s rating session timer relies of comparing the current time to the time the timer "started". In the early phases of implementation, `System.nanoTime()` was used to get this the start and current time (1 nano second = 1^e-9 seconds). This worked great for getting the current time of the timer, but updating the timer was failing. Specifically, the Fast-Forward and Re-Wind functions, which would adjust the timer by 10 seconds, were only adjusting the timer by 2 or 3 seconds. After a great deal of time double checking the logic and testing various values, it was found that a numeric overflow was being introduced on line #6 below. This is likely a fault in the Java compiler and it incorrectly inferring `10 * 1000000000` as an int, which caps at 2 147 483

647 (`10^e10 > 2147483647`). The solution was thus to solution was thus to change to milliseconds (1 millisecond = 1^e-6), which eliminated the overflow and allowed the fast-forward and re-wind functions to work perfectly.

```
/**
 * Set "start time" to be further in the future.
 * If would go below 0 seconds, reset to 0 instead.
 */
public void reWind(int numSeconds) {
    startTime = startTime + (numSeconds * MICROFACTOR);
    if (getSecondsElapsed()  < 0) {
        startTime = System.currentTimeMillis();
    }
}
```

Feature Section

The following are some features of the application:

1. **Audio recognition to identify the current time of any media playing**

   This feature recognizes the media using voice recorder. It generates the hash table and tries to match with existing media available in the media database. On recognizing the media, the user is redirected to rate and review ratings for that media content.

2. **Start, pause, and adjust time on a timer**

   The user will be able to control the timer while rating any media at any specific moment of time. The time when the timer is adjusted will be tracked and synced with the video.

3. **Create new media for rating**

   The application allows the user to add a media entry which is not already in the database and rate it. The user can enter a YouTube URL, and the APi will fetch the video and the duration of that media.

4. **Search for existing media entries**

   The media information is stored in the database in JSON format. The user will be able to search the existing media entries to rate them. The search suggestions will be given to the user for the existing media entries. If the media is not detected through sound recognition, the user can manually search for the media.

5. **View the profile of individual media content**

   The application shows information for any media content stored in the database. The media name, type, duration, and other information is displayed. The ratings given by all users for the specific media is displayed as a graph.

6. **Graph of user's rating in media profile**

   The ratings given by individual users is stored in the database and represented in the form of a graph. The rating given on each second is reflected on the graph. The users can view this graph for each media content.

7. **View graph of aggregate ratings in media profile**

   The ratings given for a particular media content is aggregated and displayed in the form of a graph. This will give an insight of the ratings given at different time periods by a number of users. The media creaters can use this to find out the ratings of different users for their media content.

Final Project Status

The project, as of writing this report, is stable and all the submitted functionalities are working satisfactorily. We achieved all our goals for the project. During testing we identified bugs for certain edge cases and we also identified innovative ways to use the application such as using it to rate books. Should we continue this project, we intend to fix the identified bugs, improve the UI, and implement other innovative features.

Submitted Functionalities

The following are the functionalities the group submitted during the last update. All of them were completed.

- ☑ Start, pause, and adjust time on a timer
- ☑ Create new ratings for media content
- ☑ Search existing media content entries
- ☑ View the profile of individual media content
- ☑ View graph of userâ€™s rating in media profile
- ☑ View graph of aggregate ratings in media profile
- ☑ Use sound recognition to identify time indexes in select media

References

1. Lars Vogel (2017) Android Services - Tutorial (http://www.vogella.com/tutorials/AndroidServices/article.html#service_intentservices)

2. Prakash Pun (2017) Retrofitâ€" A simple Android tutorial (https://medium.com/@prakash_pun/retrofit-a-simple-android-tutorial-48437e4e5a23)

3. Jerry Zhao (2017) Android ActionBar SearchView Autocomplete Example (https://www.dev2qa.com/android-actionbar-searchview-autocomplete-example/)

4. Google (2018) Android Developer Guides (https://developer.android.com/guide/)

5. Stack Overflow (2018) Included in code comments.

6. Miguel Grinberg (2013) Designing a RESTful API using Flask-RESTful (https://blog.miguelgrinberg.com/post/designing-a-restful-api-using-flask-restful)

7. Miguel Grinberg (2015) Using Celery With Flask (https://blog.miguelgrinberg.com/post/using-celery-with-flask)

Project Structure

Like any Android project, this application uses the MVC paradigm:

- **Model**:
- **View**:

  activity_main.xml This view is the main page user sees when the application is launched. An image button is used to enable the feature of 'detect media on tap' with sound recognition. Also, the feature to search media is used that allows to select media in case sound recognition function does not work. An add media button is used to add media.

  activity_add_media.xml This view enables the user to add media of their choice. It contains fields such as media title, the name of the artist, duration of media, and link to URL to add media.

  activity_info.xml This view enables the user to display the media they have selected from a list of media.The fields it includes are the duration of media selected Author, type of media and year of media launched. This view also enables to navigate to Reports and Rating screen.

  media_reports_xml.xml This view is activated once media is rated. It shows user ratings and aggregate ratings of the media.

  rating_screen.xml This view allows rating media for each second. The timer runs as long as the video runs. It then gets the input received from the user based on likes and dislikes and generates graphs.

- **Controller**: Each Android activity has an `AppCompatActivity` java object associated with it, and employs one or more service. Activities are as follows

  -- MainActivity

  -- InfoActivity

  -- AddMediaActivity

  -- ReportsActivity

  -- RatingSessionActivity.

Controller & API

Each Android activity has an `AppCompatActivity` java object associated with it, and employs one or more service.

View

Each Android activity has an `XML` page associated with it.