Software Development Documentation for Rage Tracks

Version 1.0

Group Member :Paulius Dragunas, Maciej Lato, Justin Thorsen,

Sergey Popov, Simin Liu, Fang Guo

CS 428

SPRING 2014

Table of Contents

List of Figures and Tables

1. Introduction

1.1 Identification

This Software Development Documentation(SDD) establishes the plans to be used during the development process of Rage Tracks. This mobile application applies to all IOS and Android system.

1.2 Software Overview

RageTracks.com is an online community for electronic music where users can play Sound Cloud tracks as online radio. We want to create a native mobile version as a web player, and add sharing/favoring functionality so that people can share their own music and playlists with friends and the site. We've already gotten permission from Dan Burtan, the site's owner, to access and modify his MySQL database to fit our needs.

The SSD describes the organization and procedures to be used by development group of Rage Tracks. This documentation is intended to be used by the Official Website Rage Tracks(http://ragetracks.com/), UIUC Software Engineering course and app store to demonstrate the procedures of both IOS and Android application.

2. Process

2.1 Practices

In general our team followed a modified version of eXtreme Programming. Here is a brief description of how we integrated XP practices into our process.

2.2 Iterative Development

We followed the XP standards of *The Planning Game, Small Releases,* and *Sustainable Pace* fairly closely. That is, before starting our project we created a list of user stories that described the functionalities of our system. Each user story was broken down into one or more use cases and briefly developed. Each use case was then divided into one or more small programming tasks and these tasks were then given an estimated amount of time and separated into iterations. Each iteration contained about the same amount of estimated work except the last iteration in which we allotted less time in order to be able to adjust for unexpected difficulty.

2.3 Refactoring

We additionally followed the XP practices of *Simple Design,* and *Design Improvement.* Each iteration was completed with the simplest method that achieved the new functionality we intended to add. Additionally, after each iteration the code was refactored into a more suitable design in order to prepare for the next iteration and meet good coding practices.

2.4 Testing

While we did not follow test driven development, we did ensure that each iteration included tests. Specifically, we designed tests for each feature we implemented and did not consider a feature fully implemented during an iteration unless it was properly tested. Tests included both unit test, automated GUI tests, and integration testing which varied for each system (iOS and Android).

2.5 Collaborative Development

We had an extensive amount of intra-team communication. Our project was broken into two groups, the iOS and the Android group. Each group was assigned a leader and we additionally had a team manager who handled the documentation of our progress, wiki management, and helped with communication between teams. All user interface decisions, user stories, and use cases were made by both teams in collaboration, as we wanted as close to a single user experience as possible. Outside of that and the system design was very abstractly defined and then implemented separately for each team due to the difference in systems and programming languages. Due to this fact there is not a sense of collective code ownership. We did not strictly follow *Pair Programming* and *Whole Team* as our schedules only allowed so much physical collaboration. However, we did ensure that each and every line of code was reviewed by at least two pairs of eyes. To this extend each member of each team has a generic understanding of the system.

2.6 Coding Standards

We attempted to create an overly strict set of coding standards that were to be applied to all files.

- All if/else statements MUST have brackets.
- All brackets must be on a separate line.
- All code must be auto-formatted BEFORE committing code to repository.
- All lines must be no more then 80 characters.
- No function should have more then 3 indentations (factor out nested code into other function calls)
- No function should have more then 50 lines of code.
- In line comments should be used to explain complex code.
- Function names should be descriptive but not excessively long (not longer then 30 characters)
- All functions (aside from getters/setters) must have a javadoc style commenting including a description of the function, all parameters, and return values
- Every class should have a documentation heading at the top of the file listing the author(s), and description of the class.
- For both the iOS and Android repository we will avoid branching as much as possible.
- If you need to merge a commit that cannot automatically merge, please review all code conflicts and talk with other authors to ensure not deleting essential code.
- Leave an insightful comment with each commit, and tag any partners who coded with you.

We did the best to keep code in both project to these requirements.

3. Overall Requirements & Specifications

3.1 Software Development Requirement

The project document is developed in accordance with user stories in the table below.

Table 3-1 User stories

	User stories
1	User clicks menu button, menu opens
2	User clicks menu button, menu closes

3	User swipes right, menu opens
4	User swipes left, menu closes
5	User clicks song, playing song stops, clicked song begins to play
6	User clicks pause, playing song stops playing
7	User clicks play, paused song beings playing from previous position
8	User clicks next, playing song stops, next song begins playing
9	User clicks previous, playing song stops, previous song begins playing
10	User clicks search icon, search bar opens
11	User clicks close search icon, search menu closes, all songs are loaded
12	User clicks search bar, keyboard opens
13	User clicks search icon on search bar, new songs with search key are loaded
14	User clicks a genre, new songs in genre are loaded
15	User clicks options, bottom menu bar moves to reveal waveform
16	User clicks, drags, and releases on wave form, song seeks
17	User clicks options button, bottom menu bar moves to hide waveform
18	User swipes up, songs scroll forwards, top menu disappears
19	User swipes down, songs scroll backwards, top menu appears
20	User clicks share icon, share dialog with current song is opened
21	User clicks accept in share dialog, current song is shared on Facebook
22	User clicks cancel in share dialog, share dialog is closed without sharing
23	User locks phone, user sees next/previous/play/pause controls on lock screen
24	User opens notifications, user sees current song information and controls

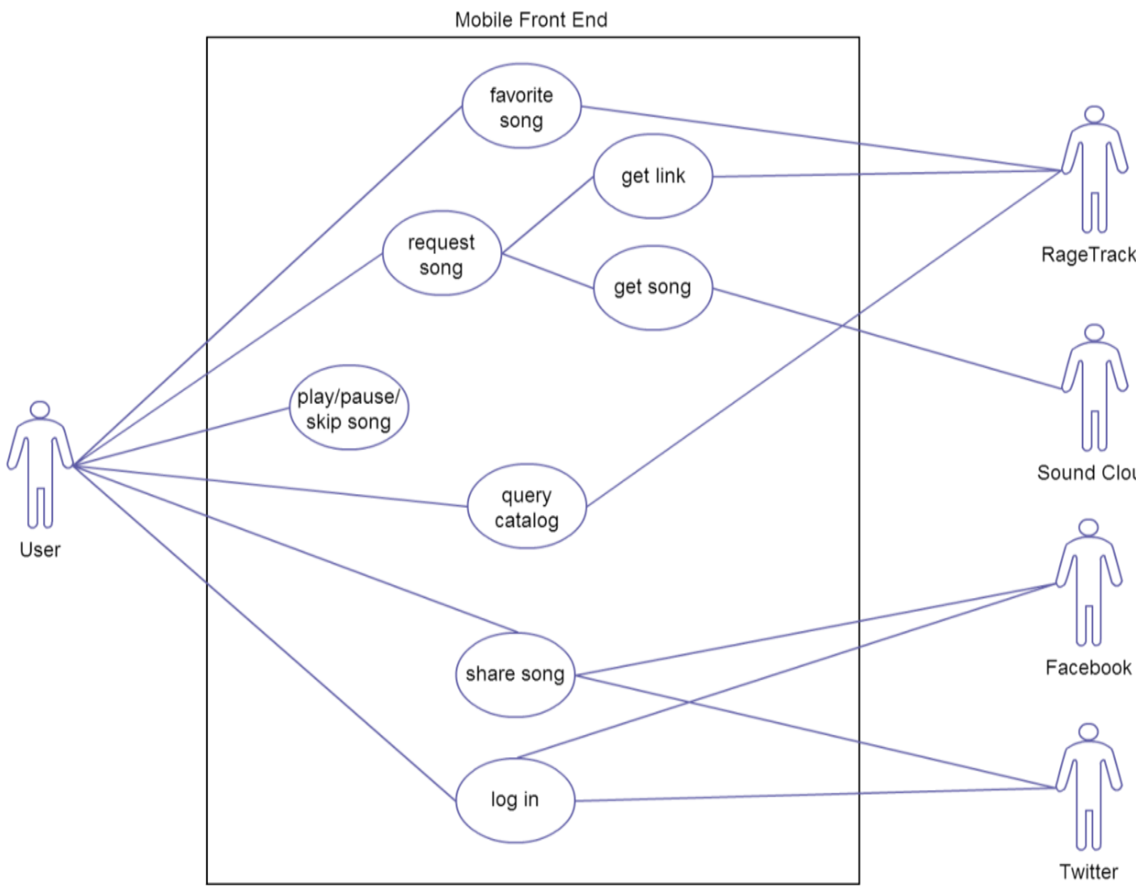
Table 3-2 Programming Tasks

Number	Programming Tasks
1	Write automated tests(Android and iOS)
2	Add menus/different genre selector/search (Android)
3	Refactor iOS app to be scalable and have good object oriented design
4	Create basic mobile player
5	Synchronize with Rage tracks song feed and display albums
6	Refactor the Cocoa tools that have been used for the iOS app
7	Finalize app controls (play, pause, next, prev) (iOS and Android)
8	Implement basic Facebook functionality (iOS and Android)
9	Improve animations (Android)
10	Add slider/seek functionality (iOS and Android)
11	Add JSON tests (Android)
12	Improve Facebook/social functionality (iOS)
13	Fix bugs in UI (iOS)
14	Add waveform seeking (Android)
15	Refactor code to use Volley and add controller tests (Android)
16	Add search functionality (Android)
17	Add lock screen player functionality(Android)

18	Add genre selection(iOS and Android)
19	Add search to the application(iOS and Android)
20	Add waveform to seek bar(iOS and Android)

3.2 Use Case Specification

Figure 3-1 Use Case Diagram



4. Architecture & Design

4.1 Design

Our system design contain a few major components: an application controller, a user-interface controller, and a music controller. Because we have two different systems: iOS and Android, we will use pseudo names for our classes and only an abstract definition of the architecture. The interface design and the actual screen are shown below.

Figure 4-1 UML Diagram 1

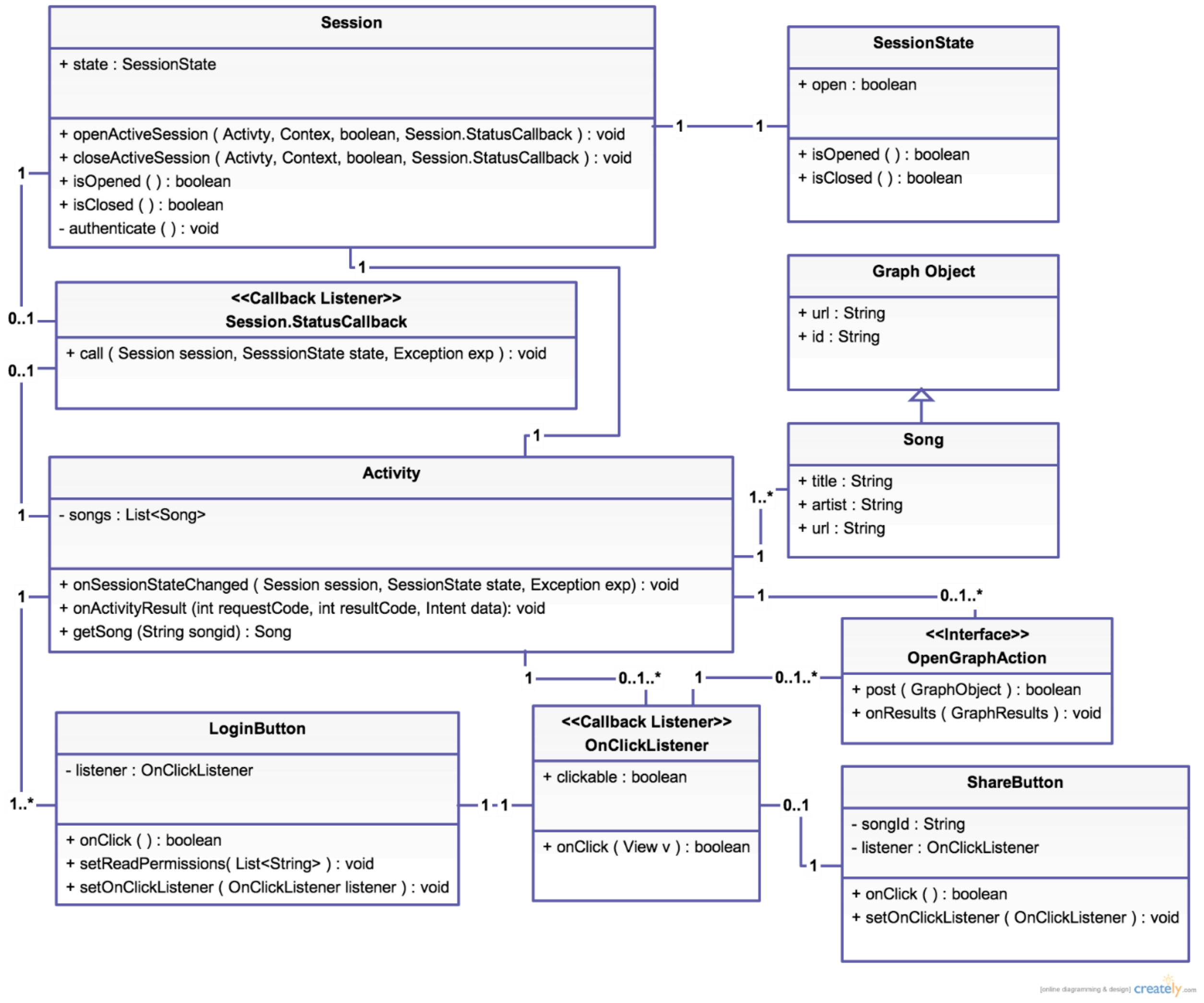
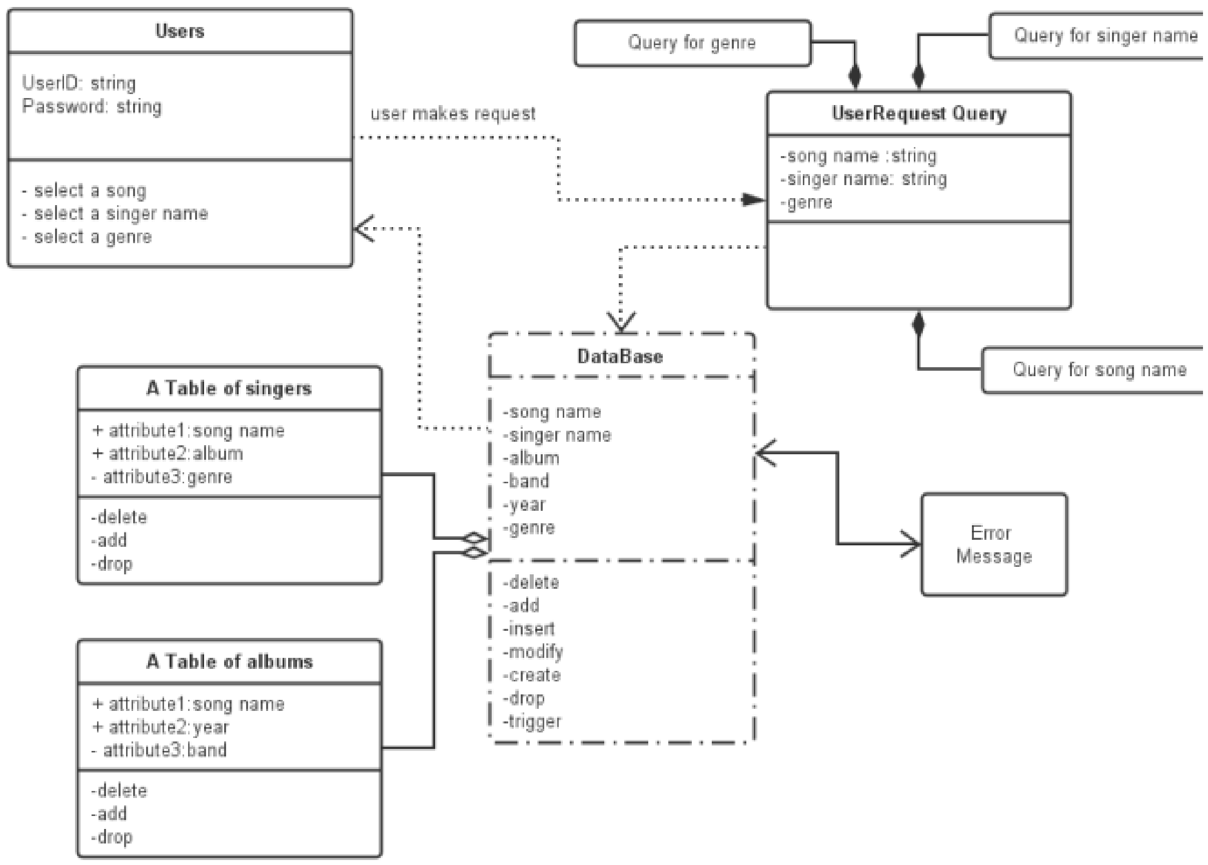

Session
+ state : SessionState
+ openActiveSession (Activty, Contex, boolean, Session.StatusCallback) : void
+ closeActiveSession (Activty, Context, boolean, Session.StatusCallback) : void
+ isOpened ( ) : boolean
+ isClosed ( ) : boolean
- authenticate ( ) : void
SessionState
+ open : boolean
+ isOpened ( ) : boolean
+ isClosed ( ) : boolean
<<Callback Listener>>
Session.StatusCallback
+ call (Session session, SesssionState state, Exception exp) : void
Graph Object
+ url : String
+ id : String
Song
+ title : String
+ artist : String
+ url : String
Activity
- songs : List<Song>
+ onSessionStateChanged (Session session, SessionState state, Exception exp) : void
+ onActivityResult (int requestCode, int resultCode, Intent data): void
+ getSong (String songid) : Song
<<Interface>>
OpenGraphAction
+ post (GraphObject) : boolean
+ onResults (GraphResults) : void
LoginButton
- listener : OnClickListener
+ onClick ( ) : boolean
+ setReadPermissions(List<String>) : void
+ setOnClickListener (OnClickListener listener) : void
<<Callback Listener>>
OnClickListener
+ clickable : boolean
+ onClick (View v) : boolean
ShareButton
- songId : String
- listener : OnClickListener
+ onClick ( ) : boolean
+ setOnClickListener (OnClickListener) : void
1
0..1
1..*
0..1..*
creately

Figure 4-2 UML Diagram 2



4.2 Application Controller

This object controls the state of the application. It contains the current song index and the list of songs. It also contains the search and category parameters. All network activities such as loading more songs, loading album art, loading waveforms are controlled through this or a member object of this class. This class has the ability to send updates to the ui controller via broadcasts. The types of broadcasts include:

```
UDATE_LOADING_SONGS
UPDATE_FINISH_LOADING_SONGS
UPDATE_ERROR_LOADING_SONGS
UDATE_LOADING_WAVEFORMS
UPDATE_FINISH_LOADING_WAVEFORMS
UPDATE_ERROR_LOADING_WAVEFORMS
UDATE_LOADING_CATEGORIES
UPDATE_FINISH_LOADING_CATEGORIES
```

```
UPDATE_ERROR_LOADING_CATEGORIES
UDATE_LOADING_SONGS
UPDATE_FINISH
```

Each of these updates is fairly self-explanatory and is broadcast to the ui controller in order to notify it of the update. The only update not obvious is UPDATE_FINISH, this update tells the ui-controller to end the application. In order to receive these updates the ui-controller must register a listener for update broadcasts.

This class also has the functions to issue commands to the music controller.

4.3 Music Controller

This class contains all of the functionality for controlling the media player on the devices. This class runs as a background service. Additionally this class has a receiver for the actions listed here:

```
ACTION_PLAY
ACTION_LOAD
ACTION_SEEK
ACTION_PAUSE
ACTION_STOP
ACTION_TOGGLE_PLAYBACK
ACTION_NEXT
ACTION_PREVIOUS
ACTION_KILL
```

Each of these commands can be broadcast to the music controller to tell it to act. Each act may have additional parameters such as the song id or the seek position. These tell the music controller what song to play/pause/load/stop/toggle or what position to seek to. The next and previous commands attempt to get the next and previous song from the Application Controller and wait for callbacks if the songs aren't loaded. The `ACTION_KILL` tells the service to stop itself and release its resources. This class also controls song states. That is, songs change to `the IDLE, LOADING, PLAYING`, or `PAUSED` state within this class. This class contains the code for both the notification available to the user and the remote lock screen controls. Both of these items act as remote views that broadcast these same commands to the music controller whenever a user clicks a button. Lastly, this class also broadcasts the following updates to the ui-controller:

```
UPDATE_PLAY
UPDATE_PAUSE
```

```
UPDATE_STOP
UPDATE_LOADING
UPDATE_POSITION
UPDATE_ERROR
```

These updates are to notify the ui controller that the song state has changed. Additionally these updates contain the song id of the song whose state has changed. This class is able to broadcast messages separately from the application controller.

4.4 User-Interface Controller

This class contains all of the controls for user interaction. It has a method for every user interface action including but not limited to:

Click on: next / previous / play / pause / song / genre / options / menu / search icon / close search / commit search / share

Swipe: left / right / up / down / seek

Each of these actions has its own function within this class and they each effect the user interface visible views or send commands through the application controller to the music controller. Additionally this controller has a receiver to receive updates from the application controller and music controller. Each of these updates notifies the user interface that it must change. For example an UPDATE_LOADING_SONGS update will tell the user interface to display a loading bar, while `UPDATE_FINISH_LOADING_SONGS` will tell it to remove the loading bar.

4.5 Framework

Our system runs on both an iOS and an Android framework. That said, the implementation details are not the same. The message passing between controllers is different in each system and the location of data may also vary. However since both frameworks are for user interaction driven applications they both required a user interface controller that handles almost all user actions. Both systems also have the ability to run background services that can toggle their state, allowing for different states of resource allocation. These two ideals in both systems lead to the core of our design. The message system, song data, and application controller ideas discussed above are only an abstract of the real implementation, as both systems did various things differently because of their unique API's and programming languages.

5. Future Plans

Justin Thorsen (thorsen1)

The project wasn't bad. In hindsight, some of our initial design choices were poor, but only because some of the Android features had a specific method of implementation they were tailored to. Oh well, that's what refactoring was for right? All in all, I'm satisfied with our final result and it is nice to now own an Android license and having gained the experience of publishing an application.

Paulius (draguna1)

I designed the original iOS Ragetracks.com app years ago. It has been of considerable success in the past years and there has been great demand for an Android version. In addition to that, I thought it would be convenient to take the CS428 project as a chance to focus on good design principles and proper programming practices. Since the design and functionality of the app was implemented once, I was able to approach it again with a more stringent set of guidelines and truly practice good programming and teamwork. I have a moderate interest in pursing this app and other iOS apps further in the future.

Maciek (lato2)

I was first approached by Dan Burtan, owner of Ragetracks.com, to design a mobile version of Rage tracks a year ago. When this class came around, I considered it the perfect opportunity to go ahead and put a team together to make this mobile app. I had some prior experience with Android development, and once I got Paulius and Justin on board I knew we had the potential to make a really great multi-platform mobile app. It was great to work with a team of students like in CS427, except this time we were much more engaged in the project (noone was really interested in Jenkins last semester). I learned a lot of cool Android tricks from Justin, and got into the habit of writing unit tests for mobile devices, which I hadn't done before. I also got to dabble around with the iOS code, which is something I've been wanting to do for a while. Learning from this team has been a great experience, and I'm excited to take what I've learned in CS428 and apply it to my career post-graduation.

Sammi (sliu19)

I was quite interested when I first heard about the project. This is my first time working with Android development, but everything works perfectly fine with help of

Maciek and Justin. They saved me hours and hours of exploring in wrong direction and guided me some fantastic functionality that sounds impossible in the beginning. The product works more than what I expected and I appreciated everyone's hard work.

Fang (fangguo1)

I was very excited to be part of the team. I did not touch iOS project before. Then this semester, after working with Paulius, I learned a lot of things from him. Not only technical but also his patient and his conscientious about the project. Even though I am still not very good at iOS right now, working in Ragetracks gives me a very good start. Also , working with other teammates benefited me a lot. So I hope during this summer I can develop my own iOS product.

Sergey (popov3)

I was very excited when my friend Maciek told me about this project. Previously, I did not have any experience in Android programming. During this class I have learned how to develop an Android application from scratch. The process we followed as a team was XP. It was an interesting process and at the same time it was easy to follow. I really liked pair programming. This valuable experience that I gained during this class will help me to develop my future projects.