Code Inspection Report

MusicBug

Client
Christopher Nguyen

Team 4
Tommy Brickwedde
Jeff Clouse
Michael Cohen
Joe DiMarino
Sean Ehrig
Kevin Wiggins

3/1/2012

Table of Contents

1. Introduction

1.1 Purpose of This Document

The purpose of this document is to provide an overview of the coding practices that were adhered to in the the creation of the MusicBug application. Practices include coding and commenting conventions as well as any software defects, in addition to a review of all systems. Lastly, meetings are outlined.

1.2 References

1. MusicBug System Requirments
2. Python (2012, Mar 2). Style Guide for Python Code. Retrieved from “http://www.python.org/dev/peps/pep-0008/”
3. MusicBug System Design Document
4. SmartBear Software (2011). White Paper: 11 Best Practices for Peer Code Review. Retrieved from “http://support.smartbear.com/resources/cc/11_Best_Practices_for_Peer_Code_Review.pdf”
5. Testing Report Document

1.3 Coding and Commenting Conventions

We adhered to camel case naming conventions for variables, methods, and classes. Classes are always signified with a starting capital letter, while instance variables are lowercase. This provides easy readability of code. Coding and commenting conventions are based off of standard Python coding conventions (Python 2012).

1.4 Defect Checklist

Defects in code can cause bugs and make it difficult to maintain. Possible defects range from logic and programming errors to missing comments. We categorized our defects into the following categories: **coding convention errors, logic errors, security oversights,** and **commenting errors**. Please view the table below, table 1, for a detailed list of defects we look for when reviewing code. Our list of found defects is found in table 2. For information about specific tests completed please see the Testing Report Document.

Table 1. Defect Categories

Category	Comments
Coding Convention Errors	Coding convention errors are when code deviates from the norm. This makes maintaining code difficult and is not good for application evolution.
Logic Errors	When code does not have the outcome originally intended.

Commenting Errors	Comments are not consistent or missing.
Security Oversights	Security oversights are vulnerabilities in code. They provide entry points for hackers and others to take advantage of your application in ways not originally intended and may inadvertently provide access to the underlying computer system.

Table 2. Defect Checklist

Category	**Defect**
Logic Error	Parameters in the wrong order
Logic Error	Incorrect Method Called
Logic Error	Array index out of bounds
Logic Error	Objects compared incorrectly
Logic Error	Incorrect parenthesising and precedence
Security Oversight	Exceptions caught inproperly
Security Oversight	Error displayed to user
Commenting Error	Missing comments
Coding Convention Error	Variable naming not best practice
Coding Convention Error	Method naming not best practice
Coding Convention Error	Class naming not best practice
Security Oversight	SQL injection vulnerability
Security Oversight	Cross-site scripting vulnerability
Security Oversight	Format string vulnerability
Security Oversight	Remote code execution
Security Oversight	Incorrect URL access restriction
Commenting Error	Too many comments
Coding Convention Error	Hard-coded as string when should be passed in as a variable

Coding Convention Error	Code duplicated instead of placed in a method

2. Code Inspection Process

2.1 Description

Code inspection was completed by the coder who wrote the code while simultaneously navigating through that part of the application. It was also inspected by a peer through Github. In future spirals code will be reviewed in pairs in person, with the coder who wrote the code paired with another coder.

Our inspection process diverged from the standard process of working with a peer in person because of conflicting schedules. Code review is a long process and can not be completed in only 1 or 2 meetings. Instead best practices state that code should be reviewed in 200 line blocks, after which it becomes much more difficult to find code defects (SmartBear 2011). That is why code was reviewed and completed remotely using Github.

2.2 Impressions of the Process

Our code inspection process has been marginally effective but could be improved with in person meetings. However, features and defects have been added to Github as issues, which helped to keep track of their progress and encouraged all group members to review associated code. This also provided a method for Team members to easily comment on both good and bad code.

Currently, the areas of our code that are least likely to have code defects are segments that have been reviewed by more than one person. Thus, views.py and urls.py are least likely to have remaining flaws, while art.py is more likely contain coding flaws.

2.3 Inspection Meetings

During inspection meetings code was reviewed and people not coding were updated with the latest code. Please refer to the Meeting Times table below, Table 3.

Table 3. Meeting Times

Date	Time	Attendance
2/27/2012	6:30pm-11:30pm	Michael Cohen, Sean Ehrig
3/1/2012	6:00pm-10:45pm	All Team Members
3/22/2012	6:00pm-9:00pm	All Team Members

3/31/2012	1:00pm-8:15pm	Michael Cohen, Sean Ehrig, Tommy Brickwedde, Jeff Clouse, Joe DiMarino
4/1/2012	2:30pm-8:00pm	Michael Cohen, Sean Ehrig, Tommy Brickwedde, Jeff Clouse, Joe DiMarino
4/29/2012	10:00am-5:00pm	Michael Cohen, Sean Ehrig, Tommy Brickwedde, Jeff Clouse
4/30/2012	5:30pm-11:00pm	Michael Cohen, Sean Ehrig, Tommy Brickwedde, Jeff Clouse
5/1/2012	5:30pm-10:00pm	Michael Cohen, Sean Ehrig, Tommy Brickwedde, Joe DiMarino, Jeff Clouse

3. Modules Inspected

MusicBug is a python based application written with the Django framework. Django folllows the Model view controller architecture and breaks the code up into three segments. Currently code exists to handle url paths as well as the controller logic for each view.

Please refer to the table below to view files with associated brief description of their functionality.

Table 4. Code Description Table

#	File	Brief Description
1	music/urls.py	The URL declarations for this Django project. Maps url paths to views. Django concept.
2	music/settings.py	The settings and configurations for our Django based project.
3	music/model.py	A class abstraction layer that wraps around each database table.
4	music/views.py	Acts as the controller in Django's model view controller framework.

5	music/templates/ base.html	The base template, extended by all other templates to include the content that is necessary for every page or view.
6	music/templates/ 404.html	Acts as a custom 404 error page.
7	music/templates/ artistBase.html	Template for all browsing pages. Extends base.html template.
8	music/templates/ artistDetails.html	Template for artist detail page. Extends artistBase.html
9	music/templates/ searchBase.html	Template for search pages. Extends base.html.
10	music/templates/ albumDetails.html	Template for album and track pages. Extends artistBase.html
11	music/templates/ browse.html	Template for browsing, extends artistBase.html
12	music/templates/ search.html	Template for searching, extends searchBase.html
13	music/templates/ rating.html	Templates for rating and viewing current ratings for albums and tracks
14	music/templates/ trackDetails.html	Template for viewing track information
15	music/static/css/ styles.css	Contains all stylesheet information
16	music/templates/ review.html	Template displays reviews for album or track and the form for reviewing
17	music/static/css/img/*	Contains images used from stylesheets
18	music/static/css/imgs/ *	Contains images used for starring items
19	music/static/js/date.js	JavaScript date library
20	music/static/js/ jquery.raty.min.js	jQuery JavaScript ratings plugin

21	music/static/js/jquery-1.7.1.min.js	jQuery JavaScript Library
22	music/static/js/scripts.j	Miscellaneous JavaScript code
23	music/templates/ channel.html	Facebook Integration File
24	music/ChartLyrics.py	Chart Lyrics API Wrapper Module
25	music/migrations/*	Contains all the South database migrations
26	music/art.py	Used for album and artist artwork
27	music/MusicBrainz.py	MusicBrainz API Wrapper Module
28	music/lyrics.py	Used to rate lyrics as explicit or not
29	music/detail_views.py	Contains the detail specific views
30	music/ facebook_views.py	Contains the facebook integrated views
31	music/tests.py	Unit Tests
32	music/ ratingandreviews.py	Used for getting recent reviews and ratings
33	music/ admin_views.py	Used for review management
34	music/templates/ admin.html	Template for moderating reviews(approving and removing)
35	music/templates/ buy.html	Template for directing user to buy page
36	music/templates/ favorites.html	Template for viewing favorites

37	music/templates/ login.html	Template for logging in to the site
38	music/templates/ paging.html	Template for viewing paged lists
39	music/templates/ profile.html	Template for viewing user profile
40	music/templates/ registration.html	Template for registering an account

4. Defects

The following table reviews the defects found in our system.

Table 5. Defects

Category:Defect	Location	Comments	Fixed
Commenting Error: Too many comments, missing comments	views.py	Comments are inconsistent and over abundant.	No
Coding Convention Error: Variable naming not best practice	views.py	Variables switch off between camel case and underscore convention use	No
Coding Convention Error: Code duplicated instead of placed in a method	views.py	Logic for common querying of database models is not placed in an additional Django Manager	No

5. Appendix A – Agreement Between Customer and Contractor

The customer agrees to a *Music Social Network* system with searching, browsing and detailed meta-data capabilities. See System Requirements Specification for more information. Additional features will be provided in further development spirals.

When and if future changes to this document occur a drafted new document will be created. Both a hard and electric copy of both versions will be presented to the client for review. Upon approval, the draft will be finalized and signed off by both parties.

Client

Name _____ Date _____
Print
Name _____ Date _____
Signature

Team

Name _____ Date _____
Print
Name _____ Date _____
Signature

Name _____ Date _____
Print
Name _____ Date _____
Signature

Name _____ Date _____
Print
Name _____ Date _____
Signature

Name _____ Date _____
Print
Name _____ Date _____
Signature

Name _____ Date _____
Print
Name _____ Date _____
Signature

Name _____ Date _____
Print
Name _____ Date _____
Signature

6. Appendix B – Team Review Sign-off

This document has been collaboratively written by all members the team. Additionally, all team

members have reviewed this document and agree on both the content and the format. Any disagreements or concerns are addressed in team comments below.

Team

Name _____ Date _____
Print
Name _____ Date _____
Signature
Comments _____
_____

Name _____ Date _____
Print
Name _____ Date _____
Signature
Comments _____
_____

Name _____ Date _____
Print
Name _____ Date _____
Signature
Comments _____
_____

Name _____ Date _____
Print
Name _____ Date _____
Signature
Comments _____
_____

Name _____ Date _____
Print
Name _____ Date _____
Signature
Comments _____
_____

Name _____ Date _____
Print
Name _____ Date _____
Signature
Comments _____
_____

7. Appendix C – Document Contributions

Michael Cohen wrote the first iteration of the introduction, the defect checklist, the coding and conventions, the code inspection process, and the modules inspected accounting for 60% of this document. Kevin Wiggins completed the defect list and collaborated in writing all other sections of this document accounting for 40% of this document.