Code Inspection Report

Dekū

Client
Shawn Squire

Team 6
Boris Boiko
Raymond Chan
Gilbert Kuo
Andrew Naviasky
Jeremy Neal

4/30/2014

Table of Contents

1. Introduction

1.1 Purpose of This Document
1.2 References
1.3 Coding and Commenting Conventions
1.4 Defect Checklist

2. Code Inspection Process

2.1 Description
2.2 Impressions of the Process
2.3 Inspection Meetings

3. Modules Inspected

4. Defects

Appendix A - Coding and Commenting Conventions

Appendix B - Peer Review Sign-off

Appendix C - Document Contributions

1. Introduction

1.1 Purpose of This Document

This document serves two primary functions. First, the document contains the coding and commenting conventions of Dekū, for use by the current engineering team, as well as any future maintainers. Second, this document also serves as a log for the code inspection process and ensures that modules have been reviewed. The primary audience for this application is the Dekū engineering team, and the secondary audience is the customer, Shawn Squire.

1.2 References

This document primarily concerns coding and commenting conventions, and code inspection logs. For greater insight into the requirements of the application, consult the SRS. For a high level view of the application design, consult the SDD.

1.3 Coding and Commenting Conventions

Coding and commenting conventions for this project are language dependent. Throughout the project, Python, JavaScript, HTML, and CSS are used.

For Python, this project follows the Python PEP8 Style Guide, available at http://legacy.python.org/dev/peps/pep-0008/.

For JavaScript, this project follow the Google JavaScript Style guide, available at https://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml.

For HTML and CSS, this project follows the Google HTML/CSS Style Guide, available at https://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml.

As an extension to each of the preceding style guidelines, avoid inline comments.

1.4 Defect Checklist

Security is a main concern throughout parts of the application. While developing using an SOA architecture has prevented developer collision and improved our ability to reuse or replace code as desired, the disparate nature of the design has left several security issues worth investigating.

- The API is fully functioning, but currently lacks safeguards against malicious use. It is accessible via a console command in most web browsers. We are in the process of securing this.
    - The main concerns are the ability to delete information from the database. The plan is to whitelist GET calls, while blocking all others from external use.

- We are constantly reevaluating how much of the user's information to store in the interface. Some information is needed by the application for API calls or displaying information, while other information is needed for permissions. Most user information should be accessible to that user, but not all. Most importantly, we are still investigating how to make certain user information private while still maintaining functionality.

2. Code Inspection Process

2.1 Description

During the development process of this application, the engineering team used a rather simplistic inspection system. By hosting the project on Github, we were provided with a set of great tools for evaluating changes on a commit to commit basis. Whenever a commit was pushed to the repository, we could quickly see at a glance what had changed. Good commit messages improved the clarity of this process.

2.2 Impressions of the Process

This process was valuable in ensuring that our code functioned properly, was well-written and cohesive. Using Github also helped because it allowed each member to address concerns at any time they could. With each member of the team having different schedules to adhere to, this was beneficial.

One criticism of this process is that while each member of the group having open access to the repository, code review was only possible after code had already been pushed to the master branch. One way of handling this would have been to create a branch for every feature a group member wished to implement, and to stage all code there for review. This is a very cumbersome process. What would have been the best implementation is to have a single group member as the owner of the repository, and have all the other members make pull requests. Ideally, this person would be the testing and implementation leader, and they could then inspect any code for issues before it gets pushed to the master branch.

2.3 Inspection Meetings

During the development process, the team did not hold inspection meetings. This may have been useful, but it was not done for two reasons. First, scheduling conflicts made it very difficult to schedule adequate blocks of time necessary. Additionally, the resources that Github provides for code inspection and analysis made this an easy process to conduct over the internet.

3. Modules Inspected

- `app/`
  - `__init__.py`: Simple script that initializes the API. Pulls in dependencies, models, routes. Has remained relatively static for most of the project.
  - `cards.py`: Module containing API methods pertaining to Card models. Currently implemented methods are fully unit-tested. May require modification.
  - `users.py`: Module containing API methods pertaining to User models. Currently implemented methods are fully unit-tested. We are iteratively adding features to those models, so this file will likely require further modification.
  - `utils.py`: Module containing any helper methods for the API. Currently only has a single simple method that assigns a necessary header to requests. May be changed or removed depending on the deployment environment. Currently required for development environment.
  - `models.py`: Module containing the schema for the Card and User models. Subject to change.
- `config.py`: Module outlining the configuration of our database. Current implementation creates a sqlite database with migration. Will change to MySQL at product launch.
- `css/`
  - `bootstrap.min.css`: Required stylesheet for Bootstrap, which is used to handle basic styling for the interface. Minified and tested externally.
  - `slidebars.min.css`: Required stylesheet for the Slidebars Jquery plugin, which is used for our side menus. Minified and tested externally.
  - `style.css`: Application-specific styling. Will be minified for production.
  - `tags.css`: Temporary stylesheet for defining colors for tags given particular keywords. May be expanded or removed based on final implementation.
- `db_create.py`: Python script that instantiates a database based on the config.py file. Will likely be changed or removed at product launch when the application switches to a MySQL database.
- `db_downgrade.py`: Python script that reverts the database to a previous version. Only needed for development and testing and will be removed at product launch.
- `db_migrate.py`: Python script that maps all existing tables to adhere to new model definitions. Will be removed at product launch.
- `db_upgrade.py`: Python script that “upgrades” the database to a newer version. Only needed for development and testing and will be removed at product launch.
- `index.html`: Only HTML file used for this application. Is relatively static, serving as container for Backbone-rendered views. Currently contains Underscore templates, but the team has plans to externalize these.
- `js/`
  - `app.js`: Main javascript file that initializes the interface. Currently contains function overriding which can be moved elsewhere.

    - `collections/`
        - `deck.js`: Backbone collection for Card model. Contains URI for API call to get cards. May change at product launch.
        - `filter_collection.js`: Backbone collection for filters used in searches. May be removed due to a plugin performing the same function.
    - `lib/`: contains minified versions of the libraries used in the application. This directory may be added to as more dependencies are required.
    - models/
        - `card.js`: Front end definition of the card class. Matches with the definition in the database.
        - `filter.js`: Class defining filters used in searching.
        - `user.js`: Front end definition of the user class. Matches with the definition in the database.
    - `views/`: Contains the logic for the various Backbone views in the interface.
- `requirements.txt`: Text file outlining Python libraries used for the project. Allows developers or a build system to download and install all dependencies quickly using pip.
- `tests.py`: Main test file for the API. Currently contains comprehensive unit tests for all API functions and possible outputs.

4. Defects

- Our models completed implemented on both the backend and the front end. However, the two systems are not communicating in instances where many-to-many relationships are defined. Specifically, following users, card metadata such as how many people have marked it, added it to their deck, etc. In our implementation, this requires inserting manually into tables. The majority of our database interactions have been abstracted to models, so this has been a challenge to implement properly.

Appendix A - Peer Review Sign-off

This section of the document verifies that each team member has reviewed and agreed to the terms set forth in this document.

Boris Boiko Signature _____ Date: ___/___/___

Comments:

Raymond Chan Signature _____ Date: ___/___/___

Comments:

Gilbert Kuo Signature _____ Date: ___/___/___

Comments:

Andrew Naviaski Signature _____ Date: ___/___/___

Comments:

Jeremy Neal Signature _____ Date: ___/___/___

Comments:

Appendix B - Document Contributions

Jeremy Neal wrote and maintained this document.