TEAM X-CEL - CODE REVIEW v2

11.26.2018

—

D H A R M I K SHAH
C A L E B CHEN
B Y R O N LEUNG
R A L P H MAAMARI

TABLE OF CONTENTS

INTRODUCTION

This document will highlight our code review strategy, and also will list each members findings based on the code that they are assigned.

Changelog:

- After D4 was handed in, we reviewed our code and made the fixes that were outlined in this document
- By now, most of the changes have been taken care of, but doing a quick code inspection after the product was complete, has led to some of the following concerns (see below)
- Our code review strategy stays the same, and by the time the TA received this document, the code will have been refactored
- After our final code review, we refactored and reformatted our working code base to take care of as many of the problems that we found as we could

CODE REVIEW STRATEGY

What to look for:

- Readability and Maintainability
    - Functions that are too large (should be separated)
    - No useless code (e.g. HTML tags that do not affect anything)
    - Comments explaining complex pieces of code
    - No unnecessary comments for things that are obvious
    - Consistent naming conventions (e.g. camel case)
    - Consistent code formatting (e.g. use of whitespace)
    - Descriptive variable names (it should be clear why each variable is used)
    - Redundant code that could be made into reusable functions

- Code Design
    - The code should follow MVC principles (the view is an .ejs file [html], which in turn should call a model .js file [jQuery], which communicates to the backend API controller, which is a .js [node] file
        - Model (JS)
        - View (.ejs)
        - Controller (JS)
    - Are there files/functions that are in the application, but never called?

- Functionality
    - Check for dead code in the database access where we return and have code underneath it that will never execute
    - Are all database err checked for, and have an appropriate response when it does occur
    - No leftover code used for debugging (e.g. console.log), or at least, no unnecessary debugging code left
    - Is the user alerted when they give an input the application cannot handle
    - Is the functionality of the code easily accessed by the people that need it
- Testing
    - Tests are easy to understand (e.g, they have a well named, clear description)
    - Tests are idempotent
    - Tests have a 100% coverage (e.g, database tests should cover if the item is in the database/if it isn't, what happens if we try to access when it is not there, etc)

CODE REVIEW SUMMARY

Byron (Dharmik's code)

- Some comments only explain what has happened in the function, but does not explain what the following code actually does (e.g. comment says "we cannot be in the login page", but doesn't explain the implications of that).
- Use of whitespace is inconsistent (e.g. random blank lines, if statements and code in one line, large JavaScript objects with multiple properties in one line)
- There are still some very large function that could be split
- There are still some event handlers that don't contain any code, they could be removed to decrease clutter
- HTML code in the javascript code should be indented properly

Dharmik (Byron's code)

- Need to add comments for the process of parsing the csv file, as it is not directly evident just based on the code itself (the XLSX npm package usage)
- In the uploadFileController and generateController, you have too much code under one function. Consider breaking this up into multiple functions

Caleb (Ralph's code)

- Not fully covered with tests (i.e. react query function is not covered).
- Formatting inconsistency - There are white spaces and random indents

Ralph (Caleb's code)

- Formatting inconsistency - There are white spaces and random indents throughout the code.
- User Interface - Some elements that are clickable don't have an `onHover` CSS class making the buttons not look clickable which leads to bad User Experience.