# *Budget Management*

### Design Review for Maintenance Status Tracking Software

Authors: Anna Kim, Ali Zaidi, Na Li, Jack Long, Joon Lim, Andy Boho

### *<2020*-*03*-*06>*, version 1.1

## Document Revision History

Rev. 1.0 <2020-03-02>: initial version

Rev. 1.1 <2020-03-06>: Final version

## Summary

Here you should give a summary of your design review. This section, which should be relatively short, focuses on the recommendations you consider to be the most important.

**Note:** this is not a summary of the *project* you reviewed. Rather, it is a summary of *your review* of that project.

The design structure is relatively clear, as there is a clear divide between what the frontend and backend components are, as well as what belongs within each of these components. Thus, there are no recommendations on how to make the design clearer. The testing methods are explained clearly with specific examples of test cases. However, how the fuzz test will be conducted is not clear enough. Still the benefits of conducting each test, including the fuzz test, are clearly specified. The groups modules are clearly lined out and they account for everything that they need for their project. They don’t seem to need any external hardware and their project is very well planned out. The group seems to have a good grasp on what they need as far as external modules. Everything they need seems to meet the specs of their project. Node.js seems to be an adequate driver for mongodb. All the tasks are well explained and seem realistic. Some time estimates seem like they need improvements. Most of them are well planned due some of them seem underestimated regarding time plans. Iterations are planned well and likely to successfully finish well.

## Questions

1. Are there any inconsistencies in the design? That is, does the document contradict itself?

Heading and sub-level headings in this design have severe issues of inconsistency in the format. For example, in part 3 --- Design Details, sub-titles in Front-end and back-end, such as Node.js Modules (Libraries) Used in Backend, API Endpoints etc., should be marked with smaller font size or numbered them separately. The same problem also happens in the Implementation Plan and Testing plan.

Besides, in the heading of part 7, Security Risk Assessment, the font size of number 7 is not inconsistent with other headings at the same level.

Suggestions: Generally, there exist serious format issues in the whole design. The group needs to pay more attention to this and make headings and bodies at the same level with the same format.

1. Are there omissions in the design? That is, are there elements that are mentioned but never discussed, or obvious pieces that are missing?

In the Table of Content, under Design Details, the sub heading of Front-end on page 4 is missing. Class Diagrams and Security Risk Assessment are listed without links to the corresponding content.

In part 3 --- Design Details, the design only lists the languages the project will use but does not mention any details of how implementation of the front-end of the webpage will be done. Besides, in the database, it fails to point out the primary key(s) in each table and the relationships between different tables.

Suggestions: The Table of Content should be modified accordingly. An ER diagram is strongly suggested to address the issue of the database.

1. Are any parts of the design unclear? The standard should be that given the design document, a competent programmer can code the project. Note that “clear” does not mean that the document must be very detailed. We assume that a decent computer scientist can fill in missing details, provided the overall document is clear enough.

The design is structured as a frontend web application, with a backend powered by NodeJS, as well as several other libraries that are mentioned in the document. Overall, the structure of the application as described in the design document seems pretty clear. This project has a clear and natural divide between what should be on the front-end and back-end respectively, and it seems like the group recognized this. There were some security-related concerns with the design that are also well addressed by the group, so overall there do not seem to be any unclear parts with the design.

1. Are there technical errors in the design? Is there any statement of fact that you know is false?

You do a really good job of listing your security concerns. Because you will be writing a web based application, make sure that you are not susceptible to cookie stealing or session hijacking. I don’t know if the libraries you are using already handle it so I just wanted to confirm. I was confused about your View and Controller section. I specifically wanted to know which components of your system fall under these categories. I agree that the MongoDB falls under your model, and I would assume that the view is the actual web pages. I would also assume your Express.js server is your controller? I just wish that it was more clear as to which section each component belonged to. I also recommend instead of going through fuzz testing, you either come up with a white list of a regular expression for different accepted use inputs. Fuzz testing is great for checking for the existence of security flaws/bugs, but not much beyond that. It is also very time consuming to generate different random inputs, especially if you want to include different button clicks and UI actions into your fuzz testing. Overall though, I do not see any statements that I find to be incorrect. In my experience using Node, this seems like the standard software architecture that is followed,so I do not see any issues in design either.

1. Has thought been given to testing? How would you test this design? What, if anything, could be done to make the design easier to test?

Overall, testing methods are well planned, in order, and connected to each other. So, when an error is caught while using one testing method, it can be further tested using other testing methods to figure out the solution. Some specific testing cases are given, which helps to understand how testing will be conducted.

Fuzz testing will be a good testing method to catch any unexpected error or crush. Since fuzz testing needs a large number of random input data, it might be time consuming to randomly enter user input data until any error or crush occurs. Another option for fuzz testing method is setting a format of user input data. This option will be time efficient, but it might miss some error pattern. I believe your team will consider both options, or even a better option, for fuzz testing to successfully find any unexpected error pattern.

1. Does the design make realistic assumptions about the environment? That is, will the team have trouble getting access to important external components (e.g., specialized hardware) and are the systems the project needs to interact with suited to the purpose?

Overall, it seems that the systems are suited for the purpose of this project. The group’s back-end is powered by a mongodb driver through Node.js, which is a very common database system for many apps that have been developed. In addition, they use express.js for the web application portion of their project, which provides them with basic web application features that they will need. The mongoose API will provide them with the queries they need to store the proper data that they need to store. They seem to use systems that will suit their project very nicely. In addition, they don’t seem to need any specialized hardware.

The group seems to have a good grasp of the modules that they will be using. They’ve accounted for things such as encrypting their passwords to avoid people hacking them as well. It seems that the group has a well-thought out plan of what they need as far as external modules to implement their project.

1. Does the plan seem realistic? Are tasks at a reasonable level of granularity and is it clear what each task means? Do the time estimates seem appropriate? Do any parts of the plan seem risky in the sense that they are likely to become a bottleneck to further progress?

The overall plan seemed realistic. The subtasks are well aligned and seemed like a good actual plan they can follow. All the tasks were easy to understand after reading descriptions and acceptance criterias they provided. Acceptance criterias provided provided all the features of user interactions on each of the tasks. All of the information was well written that anyone would easily understand the functions that will be implemented in each task.

For accounts task, the create profile subtask was given 5 units in time while editing and updating profile subtask were given 2. From the experience on Spike exercise, they should have a somewhat similar time unit assigned. Similar concern was shown in tickets task where create ticket subtask was given 5 units while view ticket was given 3 and update was given 2.

Most of the functions regarding accounts and tickets are planned to be done during first and second iterations while reports, properties, and order parts are to be done on second and third iterations. Since they are implementing essential functions in the first and second iterations and adding additional features in third iteration, their plans seem well organized and likely to avoid bottleneck when they follow plan.

1. Any other comments?

I like how Implementation Plan is structured. Each programming task is clearly stated in a table, in which it includes description of the task, acceptance criteria, sub-tasks that should be accomplished for different types of users. Tables stay consistent in the format and key points are marked with bullets, which make this part coherent and easy to read.

The group’s plan seems very well thought out. They lay out what they need to do each task. They seem to know exactly what they need for the back-end, which will pay off when it times to start coding. Keep up the great work guys!