Peer Review

Feature Name: Recommendation Engine

Developer Name: Jack Pickle

Reviewed by: Phong Cao

Date Developer Submitted: March 13th, 2023

Date Review Completed: March 14th, 2023

Major Positives and Negatives:

- Positives:
    1. Detailed design that laid out all the steps needed the feature
    2. The design was made with reusability and extensibility in mind, with helper functions that are adaptable to changes in requirements and future additions
    3. Clear separation of concerns within a class; the main functionality of a class are break into smaller helper functions that are more maintainable and understandable

- Negatives:
    1. Lack of descriptive text throughout. The design relies too much on function signature as explanation instead of actually describing what a function actually does. Although the function signature maybe enough for the developer, to someone else it is confusing
    2. I'm not sure what scoreInit does. You need to elaborate on what the helper functions do.
    3. Need to show database interaction for pullRec helper function. I'm assuming you are pulling the recommended LLI from the database with this function.
    4. Some function signatures lack return type, this makes it confusing to read the design at those points. Here is the list of functions
        - reManager.getRecs
        - validateNumRecs(int numRecs): bool
        - reService.getNumRecs(userHash, numRecs);
    5. You have some confusing namings, do not abbreviate unless it is a common knowledge. What is "arr"? Does "reHash" mean you are rehashing something?

6. You need to specify log message for success outcome
7. You need to number the failure outcomes
8. Data types should be C# data types
9. “js makes a get request for LLI recommendations with ajax fetch client - “getRecommendations”” This is not true, the ajax fetch client function is just called get from the ajax client, getRecommendations would be something you implement in recommendationEngine.js
10. You need to show what is being returned to the front end if there is an error in the back end. “If response.HasError = False, return Ok(response.Output)” does not make sense in the case of failure.
11. You should list the design goals for your design

Unmet Requirements

- You need to check if the user is authorized to use your function. Inside the token that the user passes in, there will be a claim, in our case it is the user role. You need to check if the user is authorized using this claim. I recommend you do this in your manager.
- You need to check if the user has completed the user form. This is unlikely to happen, as the user is supposed to complete their user form before using the feature, but an extra check won’t hurt. You can do this in the function where you are getting the user form data from the user. If the database return null, you can error out of the function right then, you save the system from doing extra computations

Design Recommendations:

1. Add more descriptive text to a design. Everytime you make a function call, explain what is being passed into the functions. For helper functions, write a sentence that describe what the function does, so it is clear to someone who is not the developer
2. Creations of the DAO Objects and Logger at the service layer. While this works, the code can more clearly follow the dependency inversion principle of solid if the code takes in the DAO and Logger as dependencies.
3. Instead of using the DAO Objects to access the LLI data in the database, the LLIService can be used instead. This would allow for better separation of concerns.
4. Front end design lacks details. “User chooses to get recommendations” is not enough detail, you should specify how the user can do this.
5. Creating the logs at both the manager and service classes will clutter up the log table, considering that the messages they log won’t be too different.
6. You need to add a lifeline in the Database layer that explains what is happening in the database with your interactions.
7. For our system currently, getting all the common public LLI and doing your recommendation based on them is fine. But let’s say the application scales up to 1 million users, this approach would not be scalable, as there would be heavy performance issues trying to parse through all the LLIs. To remedy this, I suggest you add some sort of filter to the LLI that you are getting, maybe limiting it to only x amount of LLI per categories, so that the system won’t suffer the more LLI are added to the database. This would make the system more extensible

Test Recommendations:

- Most of your failure cases are errors that are untestable, as they are mostly system errors. Here are some tests that you could do
    - What if the user has no User Form data. Your function should return an error
    - What if the user has no created LLI, something should still be recommended, and your function should not break