Feature name: Discovery System
Developer name: Bryan Tran
Date developer submitted: 04/15/2023
Reviewer name: Darius Koroni
Date review completed: 04/16/2023

Major Positivities and Negativities:

Top qualities of the design:

- The design creates its own score system to account for user ratings values and user ratings count to create a select statement that can quantify a listing/showcase/collaborator's value according to the search.
    - Allowing a filter to account for a greater percentage of the score's weight also means that the importance of the query takes hold for each user.
- Design takes into account each individual database's different ratings and reuses the scoring algorithm with slight modifications depending on implementation.
    - Some features use ratings of 1-5 while other features use a rating system with no upper-bound (likes or upvotes).
    - The algorithm uses the max of the count of ratings and the max of rating average as upper-bounds for scoring.
- Collaborator feature is added to the Discovery system as a gold-plated feature.
    - Collaborator was not originally intended to be searched for but it aligns with the value of the discovery system to be able to find public collaborator profiles.

Top flaws of the design:

- The default curated view when a user navigates to the discovery page is not catered to a specific user, as a result there are two possible implementations of design which could improve the current default curated view. I have listed them in Design Recommendations as "Default curated view recommendation" below.
- Query does not allow for any mistyping or altered ordering of specific strings.
    - If a user types in a query looking for the title of a showcase, adding in a string for the description will cause the query to match neither the title nor the description. This lowers the overall score despite the query matching more total values of the showcase.

Unmet Requirements:

Use case: Search bar input

- It is unclear based on wireframe or LLD whether or not the search bar input use-case is being met.
    - The success criteria states that suggested results must appear below the search bar but the wireframe and LLD do not explore this success case.

Minor Document and Design Recommendations:

Business rules: 50 results in wireframe

- 50 results should be returned during each query but wireframe does not make it clear that this requirement is being met.
- Since the server properly fetches 50 results, the front-end application can simply use the list of results as input for a function which formats the results and displays them as a series of rows and columns. The function will parse the results based on the feature and a mapping function will ensure it is performed on the entire list of requested curated results.
    - There are only 12, 16, and 20 results displayed for listings, project showcases, and collaborators respectively.

Backend validation for user input

- No trust should be given to the user's input, especially when the discovery system permissions are allowed for all users. This increases the overall security of the feature.
- This should be included in the manager layer with try-catch blocks to ensure the user cannot access the data store layer. Validation should check that input only has alphanumeric characters and is sanitized.
    - Malicious users will always try using a SQL injection attack to exploit vulnerabilities in database security.

Backend validation for Get{Feature} methods

- A check for validation should be performed at the manager layer for query inputs to ensure no attack is being performed on the database through SQL injections. A different type of validation check should be used in the service layer to ensure that the DTO created by the data access layer is valid and no entries are null.
- This check should be performed in the service layer immediately after returning from the data access layer. The values of the DTO should be checked for range (in the case of int), alphanumeric characters (in the case of strings), and non-null values as expected.
    - It is useful to ensure the methods created by other developers in your team work as properly intended. Trust, but verify.

Default curated list on discovery view

- Two different default curated view recommendations:
    1. The default curated view should be cached then served up for each user that navigates to the discovery page. This view can then be refreshed every X amount of hours to ensure the latest results and changes in listings/showcases/collaborators are reflected. The cached result can be stored

server-side in the AWS client and then returned as a JSON object by a front-end HTTPS request.

    a. This change could greatly decrease the amount of identical server calls that are made whenever any user navigates to the discovery page. Because it is the same for every user, storing the result and then serving it on request is an increase of the scalability of the product.

2. The default curated view can be specifically catered to each individual user based on their previous search history or previous booking history. Storing the user's past X amount of searches and then having a select statement filter for the most common string values could be a possible implementation.
    a. Personalized options will display results relevant to the user and increase the likelihood of them interacting with the default curated results. Since a query is being performed each time regardless, making it specific to each user will not greatly increase the amount of work performed by the database.
    b. NOTE: This would require restructuring of the code base since a default user would have to store previous search history locally as they would not have server-side storage. It is overall a better implementation in regards to feature value but requires a much greater amount of work by either introducing a new cookie that needs to be read only for discovery input, or expanding upon currently existing cookies.

Unmet requirement: Search bar input

- A recommendation for the search bar input would be to add a list of suggested results for the user to try upon clicking on the search bar. This can be done on the front-end with a static search suggestion that appears when a user performs a click. An on click method would call the relevant search box html element to appear with a state value that is set to disappear when the user changes the input value.
    - The list can be static and indicate the user for what type of queries would be effective such as "woodworking", "ceramics", or "painting". This would allow users to know what type of queries would be effective for returning results.

Distance based search query

- Allowing for a search based on proximity to user location, where the user specifies their current address, would allow for greater convenience to the user. It allows a user to decide which listings are within reach. A possible implementation would require a the bing mapping API, called using the microsoft rest library, to check the distances between the user provided address and the listing address, then stored as a list to be sorted before display to the user. The list can be sorted in the backend before being returned to the user.
    - Integrating a mapping API which performs lookups on the user address and listing address and returns the distance between the listings would allow for a sort by distance.
    - This would increase the overall value of the discovery system and work towards its intended purpose of finding relevant listings.

Test Recommendations:

Business rules: 50 results in wireframe

- Testing whether or not 50 results are displayed in the discovery page when there are 50, 49, and 51 total results expected.
    - This is a high priority test since it is based on the business requirements that 50 results must be displayed at a time when applicable for a search query.
    - This should be an E2E since the results are displayed on the front-end application, however the backend can also be tested for if the fetch and off-set command work as intended.

Backend validation for user input

- Test user input for SQL injection attacks such as 1=1, drop table, and other malicious input. The expected result is that the query is treated as a string and sanitized before input into the database language and execution.
    - This is critical for database security. Not checking user input validation can leave the system vulnerable to malicious actors attempting to gain access to our database.

Backend validation for Get{Feature} methods

- Test your validation function using incorrect DTO objects with DB.null inputs.
    - This is low priority since it is only checking that the functions created by fellow developers on the team work properly. In an ideal scenario there would be no issue.

Default curated list on discovery view

1. **Default curated view with cache**

- To test if a cache works properly, make a request to the method which returns the default curated view, modify the database, then make another request to the same method. If the cache worked properly, the change will not be reflected.
    - This has a low importance to the requirements and should be treated as gold-plating.
- To ensure the cache will update to reflect changes to the database, call the update cache method which will normally be set on a timer, this can be done using a mock timer. This should update the cache to reflect the new changes made to the data store.
    - This has a low importance to the requirements and should be treated as gold-plating.

2. **Default curated view personalized to user**

- To test that each user has a personalized default view when navigating to the discovery page, a test can be made using E2E testing where the user inputs a query for a specific term, then reloads the default page view. If the default page view is different from before the search term, then the test is successful.
    - This has a low importance to the requirements and should be treated as gold-plating.

Unmet requirement: Search bar input

- An E2E test can be made where a user clicks onto the search bar input and a check is made that suggestions appear below the search bar. The suggestions should be

example queries and ideally should disappear when a user types into the search bar but that is not a requirement.
    - There should be high priority for this test as it is directly related to the requirements of the client as stated by the BRD.

Distance based search query

- A test can be made where the user provides their current address and selects a radius of X miles to search for a listing. If the expected listings appear based on the test address provided, then the test is working.
    - This has the lowest importance to the requirements and should be treated as gold-plating since it requires a new technology to implement.