Feature name: Collaborative System
Developer name: Darius Koroni
Date developer submitted: 4/15/2023
Reviewer name: Tien Nguyen
Date review completed: 4/16/2023

Major Positivities and Negativities

Top Qualities

1. Generally, the design stays closely to the requirements.
2. Relational table design includes data types of the fields, which is very clear. This standalone design contains the exact information needed to create a database without spending so much time reading the rest of the document to figure out what type of data is needed.
3. Low Level Design is very thorough and covers all requirements from BRD.

Top Design Flaws

1. High Level Design:
    a. Unclear about which Data Access, which Data Store involved in the activity flow. This is fortunately cleared up in LLD.
    b. Confusion in Edit Collaborator Profile use case, upload and delete files appeared to be in different layers.
    c. Missing HLD of: Vote/Unvote a Collaborator, Remove Collaborator Profile. Those are covered in LLD and wireframe, indeed.
2. Relational tables: Large tables with many fields. This will cause problems for maintenance and data integrity.
3. UserVotes relational table design is different from LLD's usage. The table design doesn't have a Vote:bool column.

Unmet Requirements

1. Missing Failure case for Create Collaborator Profile in LLD.

Minor Document and Design Recommendations

1. High Level Design:
    a. Explicitly show the design uses Collaborator Data Access to access data from Collaborators Data Store
    b. Stay consistent in service calls. If the Service layer takes care of uploading files, it should also take care of delete files. Manager layer can make calls to FileService to upload or delete files. Developer can keep

Edit Collaborator Profile HLD more general such as “Update Collaborators Model”. Then go into specific use cases in Low Level Design to avoid confusion.

2. Relational Tables:
    a. The Collaborators table design allows User’s name to be different from Collaborator’s name. This will lead to inconsistent data. It’s better to pull all User’s related data from UserProfiles. Otherwise, Collaborators can have DisplayedName or Nickname field to keep them clear.
    b. Profile pictures can be stored in a separate table. Reason: Other features will just need to pull a single profile photo to display and link with the profile. If the Files table is corrupted, User can still have 1 profile photo to represent their collaborator profile.
    c. For a similar reason, a separate Expertises table would be better. Another reason is developers can have a collection of common tags to suggest for users to avoid typos, and support searching by tags in the future even though Tag is optional and less significant in this feature.
3. With assumption that LLD is the final design decision; however, UserVotes with 2 columns would be fine. 1 User can only vote once for a Collaborator. So when Manager Layer calls Vote() in Service Layer, DAO can just insert it into the database. If the insert fails, User has already voted. We can process that in the DAL. We also don’t need to store when User removes an upvote from a Collaborator Profile. We can simply delete the row from the UserVotes table.

Test Recommendations

1. Verify if a User can only vote for a Collaborator Profile once.
2. Test Collaborator Profile’s Visibility: to make sure GetCollaborator() meets business requirement
    a. If User hides the Collaborator Profile from the public, they still can edit and preview their Collaborator Profile.
    b. Another Authenticated User can’t view this hidden Collaborator Profile.
    c. Another Authenticated User who just voted on this Collaborator Profile will not be able to see the hidden Collaborator Profile anymore.
3. E2E tests to verify all success cases satisfy nonfunctional requirements such as complete the task within 5 seconds, view get updated accordingly, if error message displayed when system failed or when input is invalid.