

# DESIGN DOCUMENT

## Hubba - Discovery System

Version 1.1

Prepared By: Development Hell

Class: CECS 491B-05

Date: March 12, 2023

Github Repository:

<https://github.com/DevelopmentHellaHell/SeniorProject>

### **Team Leader**

Kevin Dinh

### **Members**

Garrett Tsumaki

Bryan Tran

Jett Sonoda

Tien Nguyen

Darius Koroni

# Revision History

Version	Overview	Date
1.0	Requirements Established, Draft Relational Tables	March 12, 2023
1.1	Revisit HLD and Stored Procedures	March 20, 2023

# Table of Contents


<b>Table of Contents</b>	<b>3</b>
<b>Overview</b>	<b>4</b>
<b>Requirements to Satisfy</b>	<b>5</b>
<b>High-Level Design</b>	<b>6</b>
<b>Stored Procedures</b>	<b>8</b>
Section I - Curated list	8
Section II - Search input	10
<b>Low-Level Design</b>	<b>13</b>
Relational Table(s)	13
Successful Use Case(s)	14
Failure Use Case(s)	18
<b>References</b>	<b>19</b>

# Overview

This document is intended to provide all need-to-know sources of the design of the Discovery System feature for potential cases of fixing, improving, debugging, and understanding all components of this feature. This document contains multiple abstraction levels of design, including use cases and database tables.

# Requirements to Satisfy

Taken directly from the BRD.

 DevelopmentHell\_BRD\_v.3.2

Business Rules:

- A list of multiple random listings or projects curated for the user will be initially be displayed on arrival to the discovery system view, default of 50 results are displayed
- Illegal input characters that are not alphanumeric will be filtered out upon searching using the search bar
- The search bar will have a limit to the number of characters that can be used for a search query, default is 200 characters
- Certain amount of results will be displayed at a time, default is 50 results
- Filtering search query can be sorted by
  - Workspaces
    - Location
    - Availability
  - Project Showcases
    - Reputation
    - Location

Requirements (list):

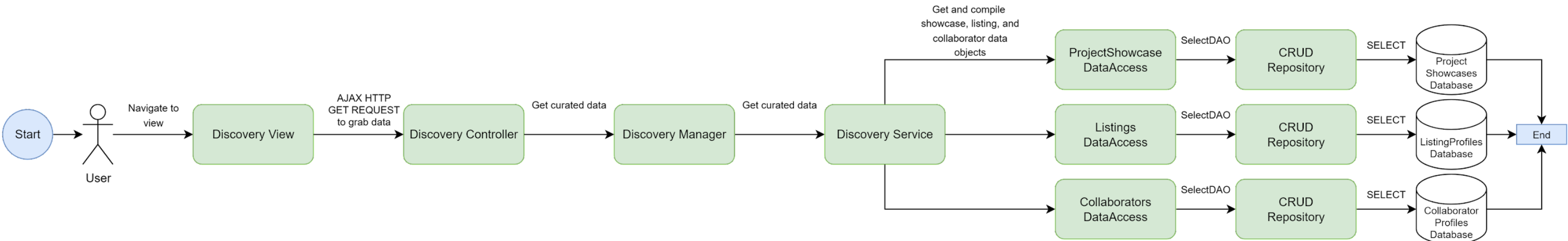
1. Search Bar Input
2. Search
3. Filter Results
4. Unfilter Results
5. Open Result

Possible gold plating:

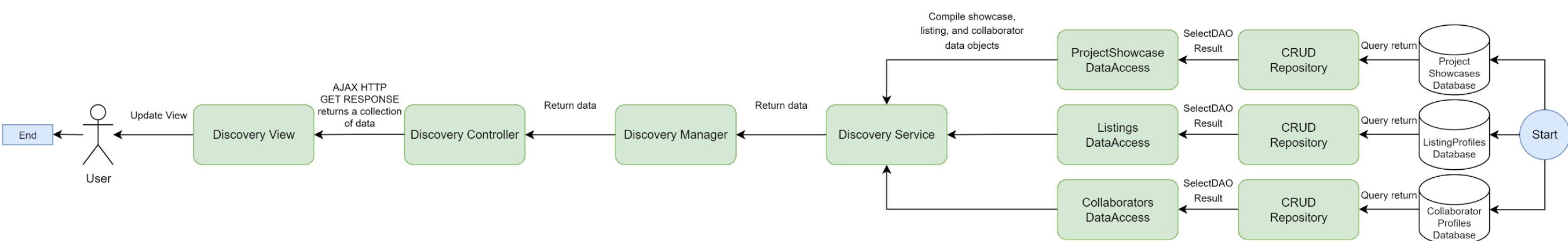
- Tune search algorithm
- Curated list from search history

# High-Level Design

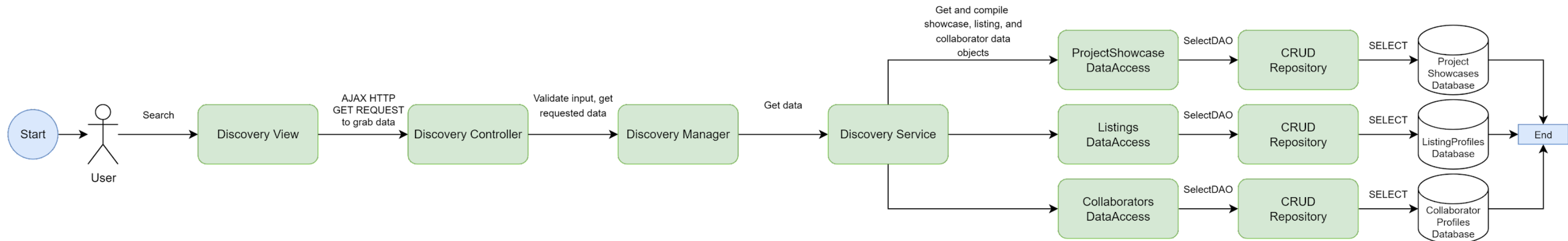
1a. Activity flow - User navigates to view



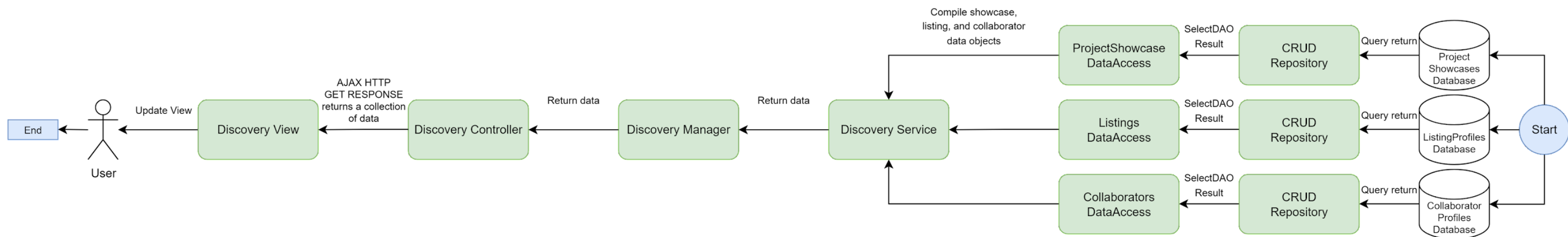
1b. Activity flow - System response



## 2a. Activity flow - Client search request (filtered/unfiltered)



## 2b. Activity flow - System response



# Stored Procedures

## Section I - Curated list

### CurateListings Procedure

```
-- CurateListings
CREATE PROCEDURE CurateListings @Offset INT
AS
SELECT L.ListingId, L.Title, L.Address, L.Price, R.AvgRatings, R.TotalRatings,
      (((ISNULL(R.AvgRatings, 0) / 5) * 0.5)                                -- RatingsRank
      + (CAST(ISNULL(R.TotalRatings, 0) AS FLOAT) / ISNULL(MAX(R.TotalRatings) OVER(), 1)) * 0.5) AS Score -- RatingsCountRank
FROM [DevelopmentHell.Hubba.ListingProfiles].[dbo].[Listings] AS L
LEFT JOIN (
    SELECT ListingId, AVG(CAST(Rating AS FLOAT)) as AvgRatings, COUNT(Rating) as TotalRatings
    FROM [DevelopmentHell.Hubba.ListingProfiles].[dbo].[ListingRatings]
    GROUP BY ListingId
) as R
ON L.ListingId = R.ListingId
WHERE L.Published = 1
ORDER BY Score DESC
OFFSET @Offset ROWS
FETCH NEXT 50 ROWS ONLY;
GO
```

- Selects relevant columns to display to the user as well as the total “Score” or rank of each result.
- The scoring/ranking algorithm is calculated by a single criteria:
  - RatingsRank - a ratio calculated by the average ratings for a particular listing divided by 5 (the maximum rating score)
  - RatingsCountRank - a ratio calculated by the total number of ratings for a particular listing divided by the top total ratings of the current search
- Returns a result of 50 listings with the highest score/rank given an offset

### CurateCollaborators Procedure

```
-- CurateCollaborators
CREATE PROCEDURE CurateCollaborators @Offset INT
AS
SELECT C.CollaboratorId, C.Name, V.TotalVotes
FROM [DevelopmentHell.Hubba.CollaboratorProfiles].[dbo].[Collaborators] AS C
LEFT JOIN (
    SELECT CollaboratorId, COUNT(AccountId) as TotalVotes
    FROM [DevelopmentHell.Hubba.CollaboratorProfiles].[dbo].[UserVotes]
    GROUP BY CollaboratorId
) as V
ON C.CollaboratorId = V.CollaboratorId
WHERE C.Public = 1
ORDER BY V.TotalVotes DESC
OFFSET @Offset ROWS
FETCH NEXT 50 ROWS ONLY;
GO
```

- Returns a result of 50 collaborators with the highest votes given an offset



### CurateShowcases Procedure

```
-- CurateShowcases
CREATE PROCEDURE CurateShowcases @Query NVARCHAR(200)
AS
SELECT S.Id, S.Title, S.Rating
FROM [DevelopmentHell.Hubba.ShowcaseProfiles].[dbo].[Showcases] AS S
WHERE S.IsPublished = 1
ORDER BY S.Rating DESC
OFFSET @Offset ROWS
FETCH NEXT 50 ROWS ONLY;
GO
```

- Returns a result of 50 showcases with the highest ratings given an offset

## Section II - Search input

### SearchListings Procedure

```
-- SearchListings
CREATE PROCEDURE SearchListings @Query NVARCHAR(200), @Offset INT, @FTTableRankWeight FLOAT, @RatingsRankWeight FLOAT, @RatingsCountRankWeight FLOAT
AS
SELECT L.ListingId, L.Title, L.Address, L.Price, R.AvgRatings,
       ((CAST(FT.Rank AS FLOAT) / ISNULL((NULLIF(MAX(FT.Rank) OVER(), 0)), 1) * @FTTableRankWeight) -- FTTableRank
        + ((ISNULL(R.AvgRatings, 0) / 5) * @RatingsRankWeight) -- RatingsRank
        + ((CAST(ISNULL(R.TotalRatings, 0) AS FLOAT) / ISNULL(MAX(R.TotalRatings) OVER(), 1)) * @RatingsCountRankWeight) AS Score -- RatingsCountRank
FROM [DevelopmentHell.Hubba.ListingProfiles].[dbo].[Listings] AS L
INNER JOIN FREETEXTTABLE(
    [DevelopmentHell.Hubba.ListingProfiles].[dbo].[Listings],
    (Title, Description, Address),
    @Query
) AS FT
ON L.ListingId = FT.[Key]
LEFT JOIN (
    SELECT ListingId, AVG(CAST(Rating AS FLOAT)) as AvgRatings, COUNT(Rating) as TotalRatings
    FROM [DevelopmentHell.Hubba.ListingProfiles].[dbo].[ListingRatings]
    GROUP BY ListingId
) as R
ON L.ListingId = R.ListingId
WHERE L.Published = 1
ORDER BY Score DESC
OFFSET @Offset ROWS
FETCH NEXT 50 ROWS ONLY;
GO
```

- Selects relevant columns to display to the user as well as the total “Score” or rank of each result.
- The scoring/ranking algorithm is calculated by the 3 different variables:
  - FTTableRank - a ratio calculated by the rank score given from the FREETEXTTABLE function in SQL that is divided by the top rank of the current search
  - RatingsRank - a ratio calculated by the average ratings for a particular listing divided by 5 (the maximum rating score)
  - RatingsCountRank - a ratio calculated by the total number of ratings for a particular listing divided by the top total ratings of the current search
- Each rank ratio is multiplied by weights that adds up to 1 which determines the how much each variable affects the search (Default values are 0.5, 0.25, 0.25 respectively).
- Returns a result of 50 listings with the highest score/rank given an offset, query, and the following weight parameters

## SearchCollaborators Procedure

```
-- SearchCollaborators
CREATE PROCEDURE SearchCollaborators @Query NVARCHAR(200), @Offset INT, @FTTableRankWeight FLOAT, @VotesCountRankWeight FLOAT
AS
SELECT C.CollaboratorId, C.Name, V.TotalVotes
      ((CAST(FT.Rank AS FLOAT) / ISNULL((NULLIF(MAX(FT.Rank) OVER(), 0)), 1) * @FTTableRankWeight -- FTTableRank
      + ((CAST(ISNULL(V.TotalVotes, 0) AS FLOAT) / ISNULL(MAX(V.TotalVotes) OVER(), 1)) * @VotesCountRankWeight) AS Score -- VotesCountRank
FROM [DevelopmentHell.Hubba.CollaboratorProfiles].[dbo].[Collaborators] AS C
INNER JOIN FREETEXTTABLE(
      [DevelopmentHell.Hubba.CollaboratorProfiles].[dbo].[Collaborators],
      (Name, ContactInfo, Tags, Description, Availability),
      @Query
) AS FT
ON C.CollaboratorId = FT.[Key]
LEFT JOIN (
      SELECT CollaboratorId, COUNT(AccountId) as TotalVotes
      FROM [DevelopmentHell.Hubba.CollaboratorProfiles].[dbo].[UserVotes]
      GROUP BY CollaboratorId
) as V
ON C.CollaboratorId = V.CollaboratorId
WHERE C.Public = 1
ORDER BY Score DESC
OFFSET @Offset ROWS
FETCH NEXT 50 ROWS ONLY;
GO
```

- Selects relevant columns to display to the user as well as the total “Score” or rank of each result.
- The scoring/ranking algorithm is calculated by the 2 different variables:
  - FTTableRank - a ratio calculated by the rank score given from the FREETEXTTABLE function in SQL that is divided by the top rank of the current search
  - VoteCountRank - a ratio calculated by the total number of votes for a particular collaborator divided by the top total votes of the current search
- Each rank ratio is multiplied by weights that adds up to 1 which determines the how much each variable affects the search (Default values are 0.5, 0.5 respectively).
- Returns a result of 50 collaborators with the highest score/rank given an offset, query, and the following weight parameters

## SearchShowcases Procedure

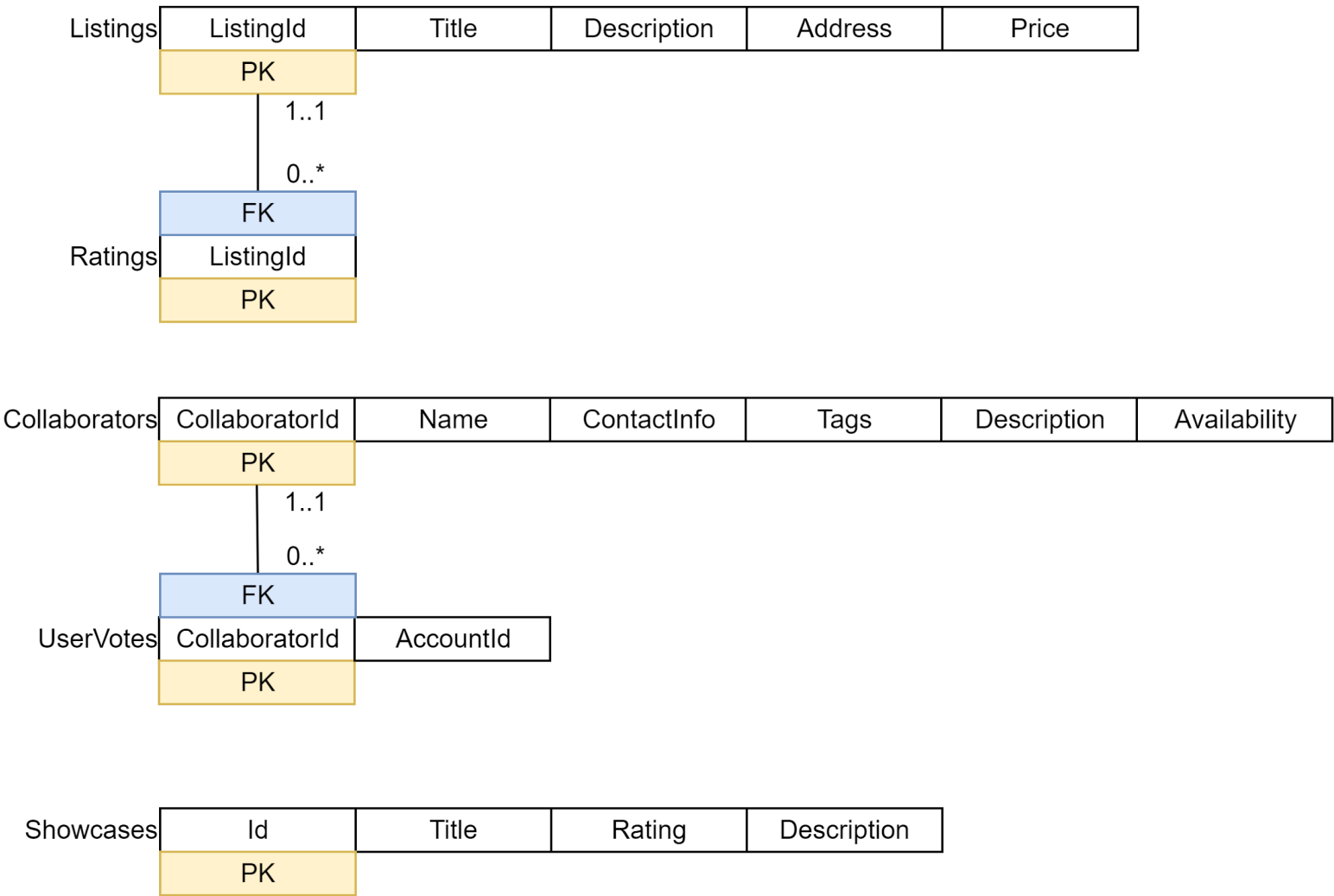
```
-- SearchShowcases
CREATE PROCEDURE SearchShowcases @Query NVARCHAR(200), @Offset INT, @FTTableRankWeight FLOAT, @RatingsRankWeight FLOAT
AS
SELECT S.Id, S.Title, S.Rating
      ((CAST(FT.Rank AS FLOAT) / ISNULL((NULLIF(MAX(FT.Rank) OVER(), 0)), 1) * @FTTableRankWeight -- FTTableRank
      + ((CAST(ISNULL(Ratings, 0) AS FLOAT) / ISNULL(MAX(Ratings) OVER(), 1)) * @RatingsRankWeight) AS Score -- RatingsRank
FROM [DevelopmentHell.Hubba.ShowcaseProfiles].[dbo].[Showcases] AS S
INNER JOIN FREETEXTTABLE(
      [DevelopmentHell.Hubba.ShowcaseProfiles].[dbo].[Showcases],
      (Title, Description),
      @Query
) AS FT
ON S.Id = FT.[Key]
WHERE S.IsPublished = 1
ORDER BY Score DESC
OFFSET @Offset ROWS
FETCH NEXT 50 ROWS ONLY;
GO
```

- Selects relevant columns to display to the user as well as the total “Score” or rank of each result.
- The scoring/ranking algorithm is calculated by the 2 different variables:
  - FTTableRank - a ratio calculated by the rank score given from the FREETEXTTABLE function in SQL that is divided by the top rank of the current search
  - RatingsRank - a ratio calculated by the ratings for a particular showcase divided by the highest total rating for the current search
- Each rank ratio is multiplied by weights that adds up to 1 which determines the how much each variable affects the search (Default values are 0.5, 0.5, respectively).
- Returns a result of 50 showcases with the highest score/rank given an offset, query, and the following weight parameters

# Low-Level Design

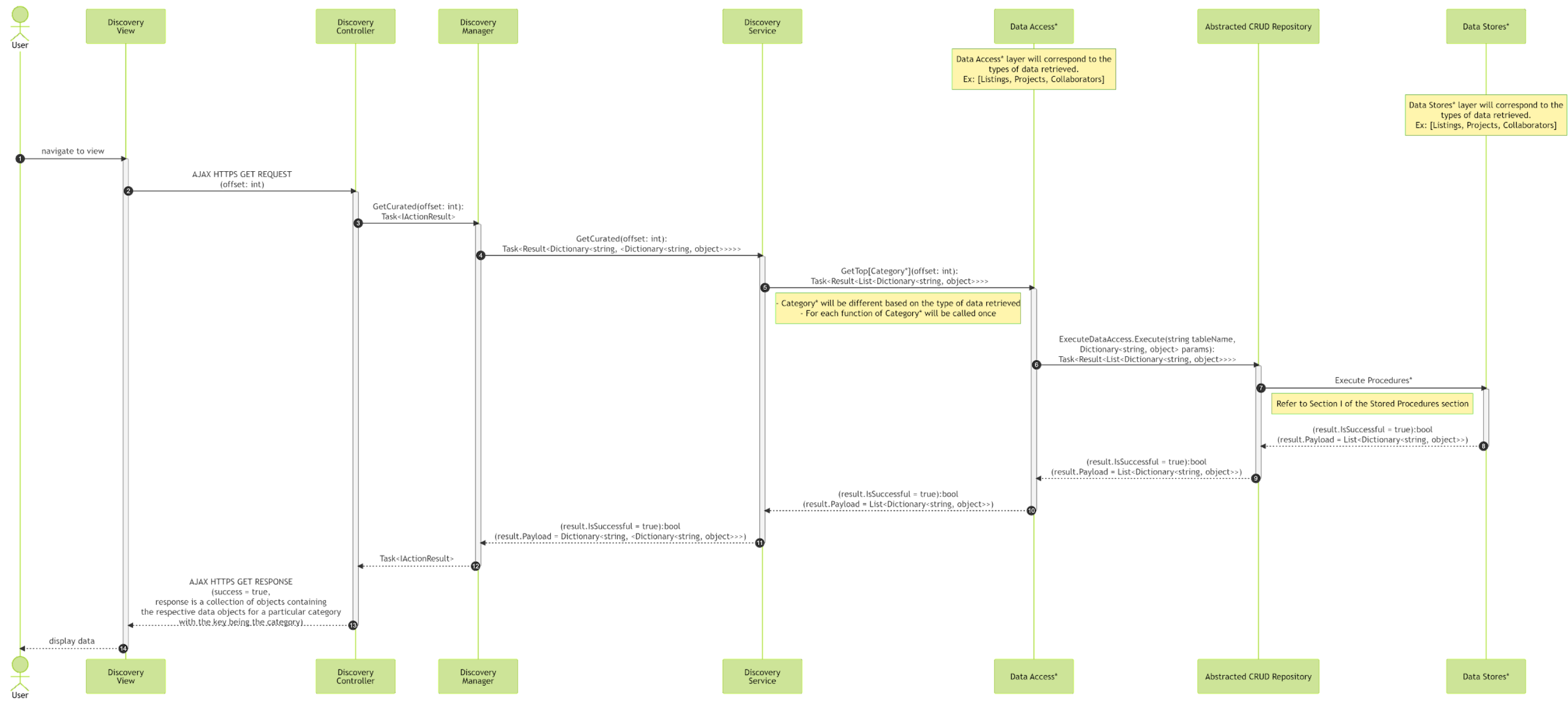
## Relational Table(s)

The tables shown do not include all columns. Only the ones that are used for the discovery system are visible.



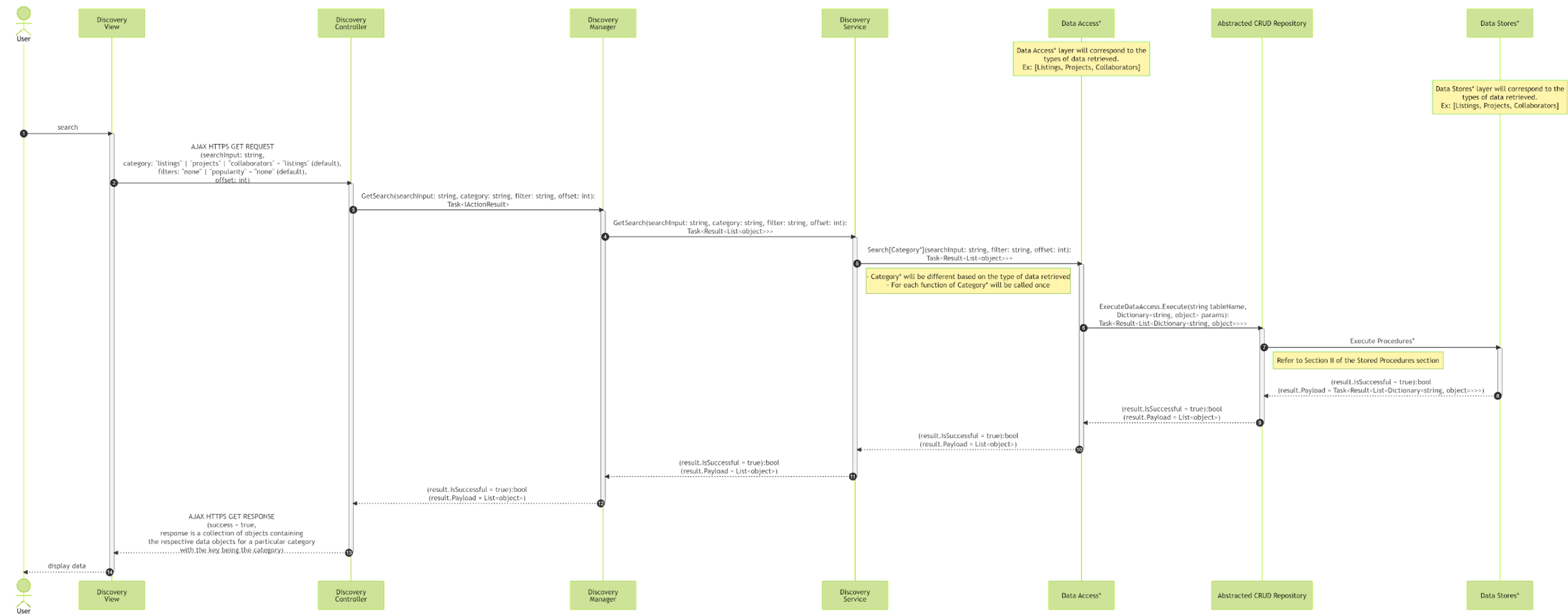
# Successful Use Case(s)

A list of multiple random listings or projects curated for the user will be initially be displayed on arrival to the discovery system view, default of 50 results are displayed

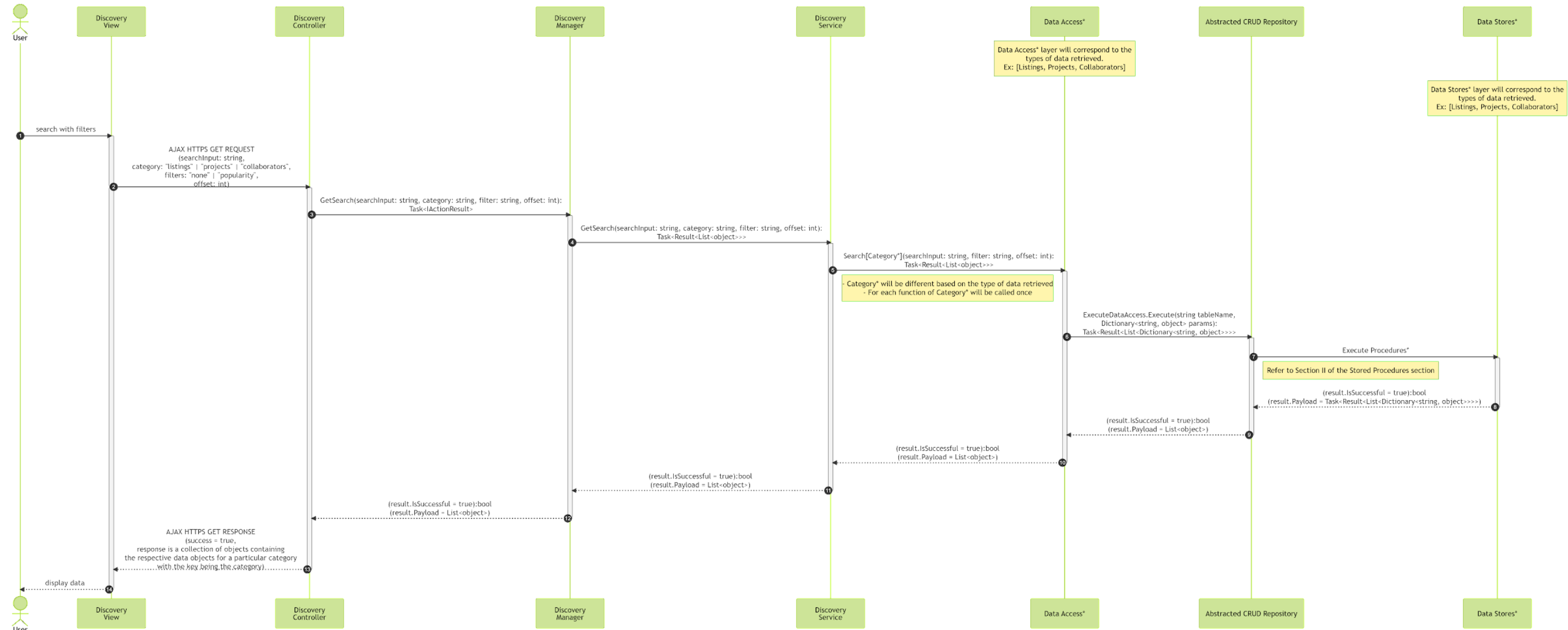


Search / Unfilter Results (Search with no filters)

- A regular search is a search with no filters applied.
- Unfiltered results is an action performed by the actor on the front end (Discovery View) and will use the default parameters to unfilter results.

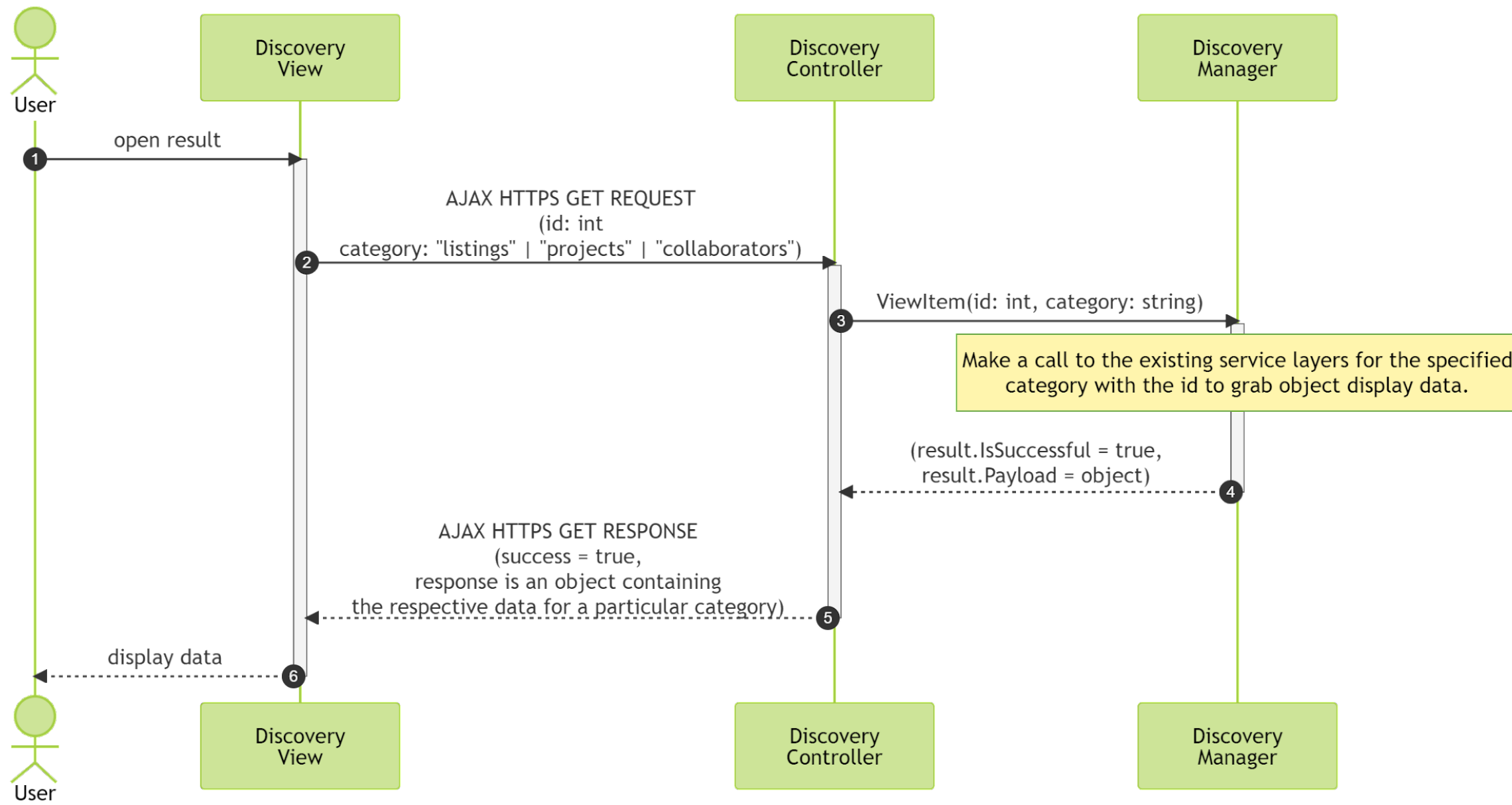


Filter Results



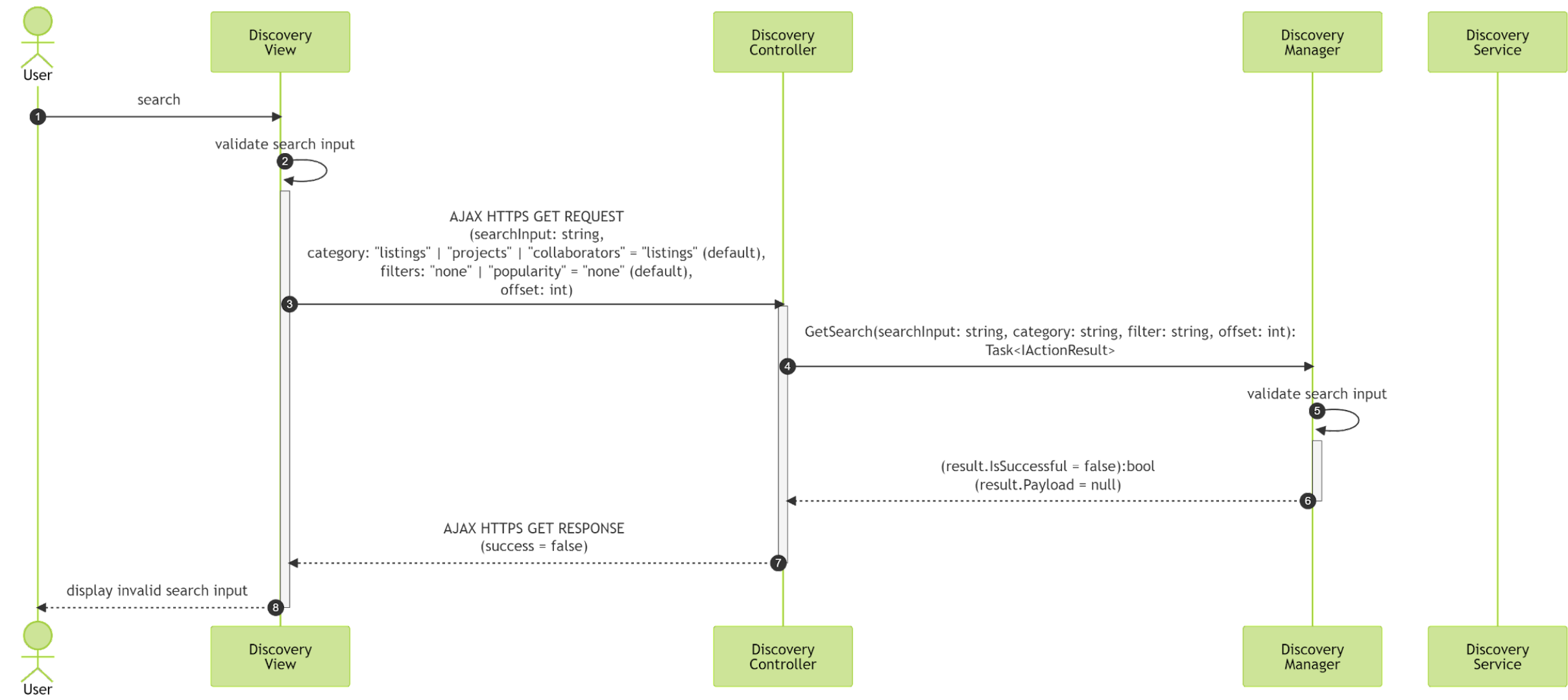


# Open Result

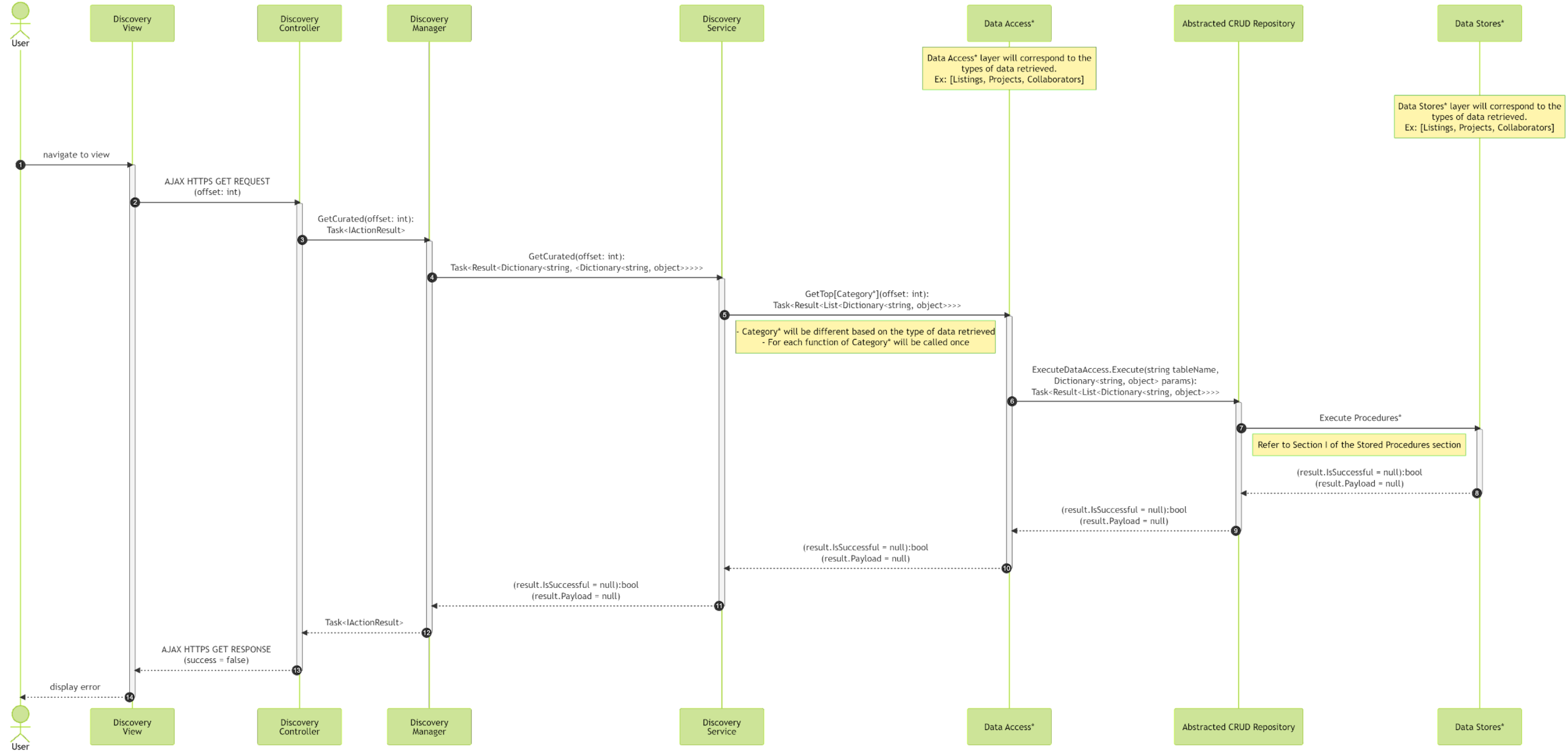


Failure Use Case(s)

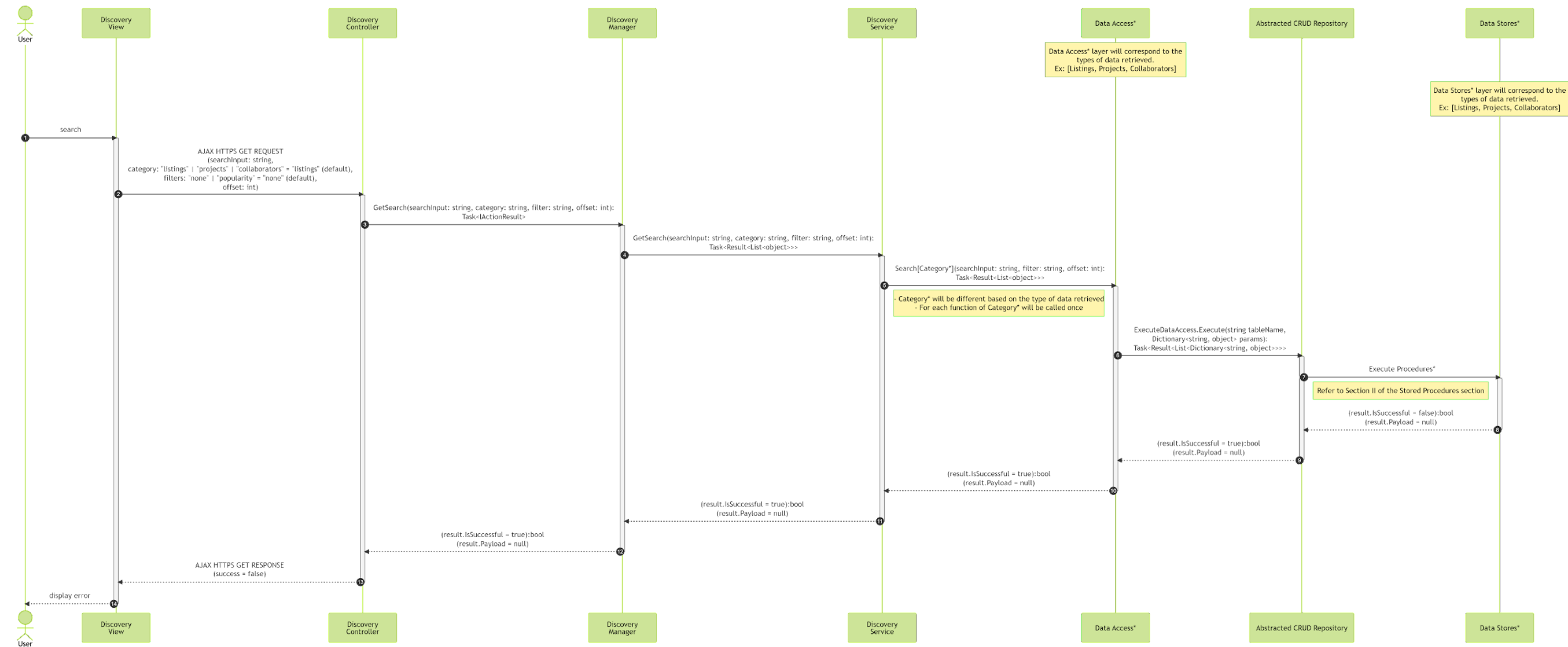
Input is invalid



Database error - Curated Data



Database error - Search Data



## References

This document elaborates client's requirements<sup>1</sup> for one of the product's core components - DISCOVERY SYSTEM.  
The requirement is designed based on the system architecture provided in High-Level Design Document<sup>2</sup>.  
The diagrams included in this document utilized diagrams.net<sup>3</sup> and Mermaid.js<sup>4</sup>.

---

<sup>1</sup> Client's email, 10/17/2022

<sup>2</sup> High-Level Design Document, URL: <https://github.com/DevelopmentHellaHell/SeniorProject/blob/b43182c4076d471ef03520052675f1f88371dafd/docs/HL%20Design/DevelopmentHell%20HLD%20v1.2.pdf>

<sup>3</sup> <https://www.diagrams.net/>

<sup>4</sup> <https://mermaid-js.github.io/mermaid/#/>