Table of Contents:

Data Types:

User

Attributes	**Data Types**	**Nullable**
FirstName	String	Not Null
LastName	String	Not Null
Nickname	String	Not Null
Email	String	Not Null
Password	String	Not Null
PhoneNumber	String	Null
PhoneType	String	Null
Share	String	Null

Item

Attributes	**Data Types**	**Nullable**
ItemID	Integer	Not Null
Name	String	Not Null
Description	String	Null
Condition	String	Not Null
Type	String	Not Null
Platform	String	Null
Media	String	Null
PieceCount	String	Null

Table of contents

Swap

Attributes	**Data Types**	**Nullable**
ProposerEmail	String	Not null
CounterpartyEmail	String	Not null
ProposedItemID	Integer	Not null
CounterPartyItemID	Integer	Not null
SwapID	Integer	Not null
Status	String	Not Null
ProposedDate	Date	Not Null
AcknowledgedDate	Date	Not Null
ProposerRating	Integer	Not Null
CounterPartyRating	Integer	Not Null

Location

Attributes	**Data Types**	**Nullable**
State	String	Not Null
City	String	Not Null
PostalCode	String	Not Null
Latitude	Float	Not Null
Longitude	Float	Not Null

Table of contents

Business Logic Constraints:

GameSwap User

- New Gameswap User must register first.
- The user who has an existing account will not be able to register.
- The user cannot swap items with themselves.

GameSwap Swap

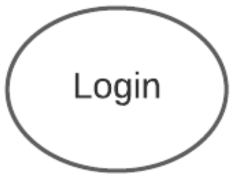
- Item is not available for swapping if the swap request is pending or completed.
- Acknowledged_date is NULL until the counterparty accepts the swap request.
- If a swap request is rejected, then the same Item-to-Item cannot be requested for swap again.
- Once a swap is complete, we cannot swap the same item again. We can, however, add the same ITEM again: which will now have a new ITEMID and make it available for swapping.
- Proposer contact information(if available) is visible to the counterparty only after swaps are accepted.
- Swap between users is marked completed only after they rate each other.
- The rating scale is between 0-5.

GameSwap Item

- Users will be allowed to add items only if the ratings on hold are not greater than 2 and the number of unaccepted swaps is not more than 5
- Users will be able to search for items based on either the Keyword/ Postal Code/ Miles
- If postal code for an item is not the same as the user, distance (calculated) will be shown
- User information cannot be updated if there's pending swaps

<u>Login</u>

Task Decomp



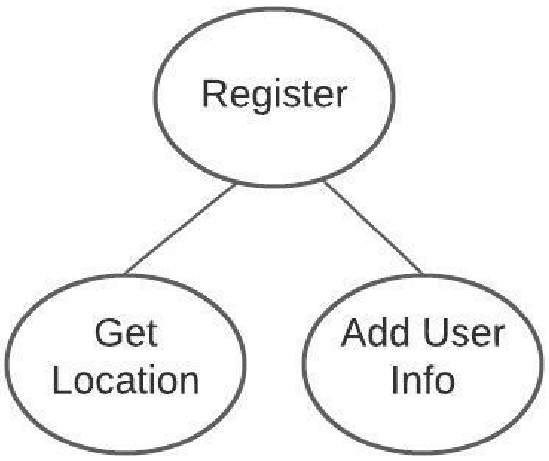
Lock Types: Read-only on User table
Number of Locks: Single
Enabling Conditions: None
Frequency: Any number of logins per day
Consistency (ACID): not critical.
Subtasks: Mother Task is not needed. No decomposition needed.

Abstract Code

- User enters *Email/phone number* (‘$Username’), *Password* (‘$Password) input fields.
- If data validation is successful for both *Email/phone number* and *Password* input fields, then:
  - When the ***Login*** button is clicked:
    - If the User record is found and User.Password != ‘$Password’:
      - Go back to the **<u>Login</u>** form, with an error message.
    - Else:
      - Store the User.Email as session variable ‘$User’.
      - Go to the **<u>Main Menu</u>** page.
- Else *Email/phone number* and *Password* input fields are invalid, display the **<u>Login</u>** form with error message.

Register

Task Decomp



Lock Types: Read from Location table and Write to User table.
Number of Locks: Two different schema constructs are needed
Enabling Conditions: None
Frequency: Same
Consistency (ACID): not critical.
Subtasks: Mother Task is needed. Order is necessary.

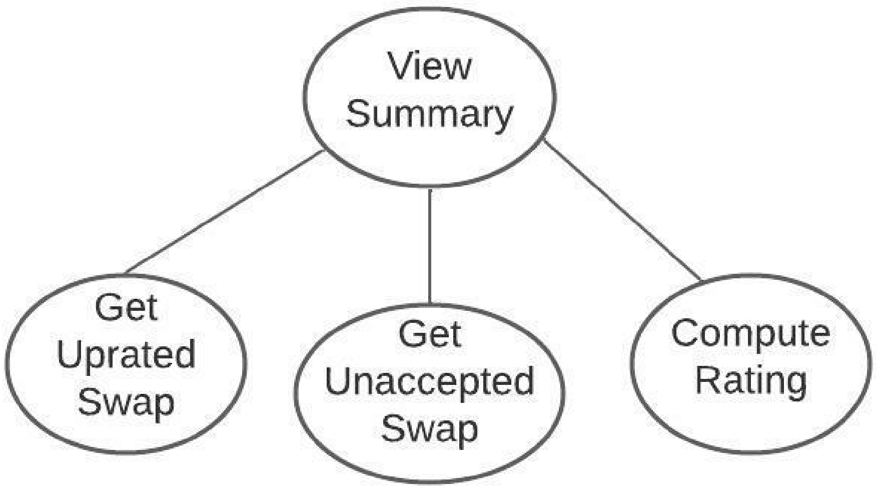
Abstract Code

- User clicked on the ***Register*** button from **Login**.
- Run the **Register** task:
    - Get the Locations; Populate the *Postal Code* ('$PostalCode') dropdown in the UI.
    - User enters *Email* ('$Email'), *Password* ('$Password'), *First Name* ('$FirstName'), *Last Name* ('$LastName'), *Nickname* ('$Nickname') input fields.
    - User selects a postal code from the *Postal Code* ('$PostalCode') dropdown, which auto-populates the *City* and *State* input fields.
    - User optionally provides phone number information:
        - User enters *Phone number (optional)* ('$PhoneNumber') input field.
        - User selects a phone number type from the *Type* ('$PhoneNumberType') dropdown.
        - User may check the *Show phone number in swaps* ('$ShowPhoneNumber') checkbox.

- If data validation is successful for all the provided fields, then:
  - When the ***Register*** button is clicked:
    - If a User record with User.Email == '$Email' exists:
      - Go back to the **User Registration** form with error message.
    - Else if a User with User.PhoneNumber == '$PhoneNumber' exists:
      - Go back to the **User Registration** form with error message.
    - Else:
      - Add a new User record for the user.
      - Store the User.Email as session variable '$User'.
      - Go to the **Main Menu**.
- Else Display the **User Registration** form with error message.

View Summary

Task Decomp



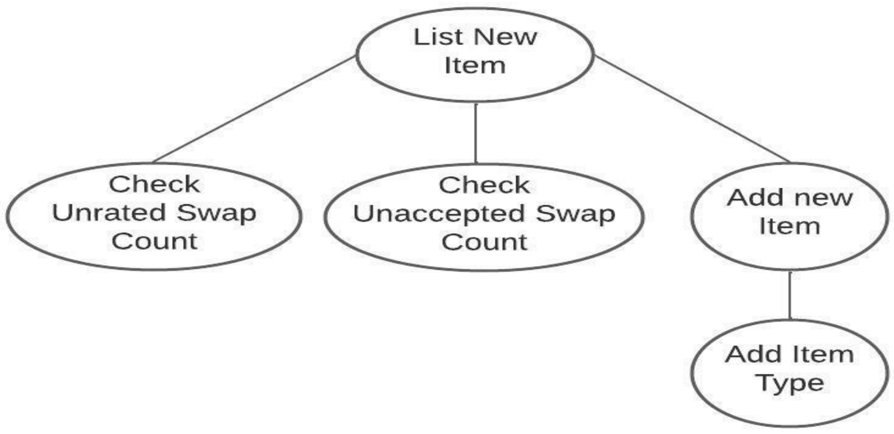
Lock Types: Read-only on User and Swap table.
Number of Locks: Single
Enabling Conditions: It is enabled by successful user's login
Frequency: Same
Consistency (ACID): not critical.
Subtasks: Mother Task is needed. Order is not necessary.

Abstract Code

- Show ***List Item***, ***My items***, ***Search items***, ***Swap history***, ***Update my info*** and ***Logout*** buttons.
- Run the **View Summary** task:
    - Find the User record for user '$User'; Display the user first name and last name in the UI.
    - Find Swap where the Swap.ProposerEmail is User.Email
        - Sum up each Swap.CounterpartyRating and divide by the number of occurrences; Display the result as *My Rating* field in the UI
    - Find Swaps where the Swap.ProposerEmail is User.Email and the Swap.Status is 'COMPLETED' but Swap.CounterpartyRating is null or find Swaps where the Swap.CounterpartyEmail is User.Email and Swap.Status is 'COMPLETED' but Swap.ProposerRating is null.
        - Sum up the occurrences; Display the result as ***UnratedSwaps*** link
    - Find Swaps where the Swap.CounterpartyEmail is User.Email and the Swap.Status is 'PENDING'.
        - Sum up the occurrences; Display the result as ***UnacceptedSwaps*** as link.
- Upon:
    - Click ***List Item*** button - Jump to **List New Item** task.
    - Click ***My items*** button - Jump to **View My Items** task.
    - Click ***Search items*** button - Jump to **Search for item** task.
    - Click ***Swap history*** button - Jump to **View Swap History** task.
    - Click ***Update my info*** button - Jump to **Update User Information** task.
    - Click ***Unaccepted swaps*** link - Jump to the **Accept/Reject Swap** task.
    - Click ***Unrated swaps*** link - Jump to the **Swap Rating Update** task.
    - Click ***Logout*** button - Clear the '$User' session variable and go to the **Login** form.

List New Item

Task Decomp



Lock Types: Write on Item table and "type-specific" table.
Number of Locks: 1 or 2 schema constructs are needed based on type.
Enabling Conditions: It is enabled when user clicks "List Item" in Main menu.
Frequency: Same
Consistency (ACID): critical. As the item added, its type needs to be updated on the type-specific table
Subtasks: Mother Task is needed. Order is necessary.

Abstract Code

- User clicked on the ***List Item*** button from **Main Menu**.
- Run the **List New Item** task:
    - Find Swaps where the Swap.ProposerEmail is '$User' and the Swap.Status is 'COMPLETED' but Swap.CounterpartyRating is null or find Swaps where the Swap.CounterpartyEmail is '$User' and Swap.Status is 'COMPLETED' but Swap.ProposerRating is null.
        - Sum up the occurrences.
        - If the number of unrated swaps > 5, then:
            - Display a popup with an error message.
            - End **List New Item** task.
    - Find Swaps where the Swap.CounterpartyEmail is '$User' and the Swap.Status is 'PENDING'.
        - Sum up the occurrences.
        - If the number of unaccepted swaps for user '$User' > 2, then:
            - Display a popup with an error message.

        - End **List New Item** task.
- Go to **<u>Listing an Item</u>** form.
    - User selects a game type from the *Game type* (‘$GameType’) dropdown:
        - Find the list of additional fields to add to the UI if any:
            - If ‘$GameType’ == “Video Game", then:
                - Show the *Platform* (‘$Platform’) dropdown and the *Media* (‘$Media’) dropdown.
                - Find the platforms for ‘$GameType’ == “Video Game" and populate the *Platform* dropdown.
                - Find the media and populate the *Media* dropdown.
            - Else if ‘$GameType’ == “Computer game", then:
                - Show the *Platform* (‘$Platform’) dropdown.
                - Find platforms for ‘$GameType’ == “Computer Game" and populate the *Platform* dropdown.
            - Else if ‘$GameType’ == “Jigsaw puzzle", then:
                - Show the *Piece count* (‘$PieceCount’) input field.
        - User enters *Title* (‘$Name’) input field.
        - User selects a condition from the *Condition* (‘$Condition’) dropdown.
        - User selects a platform from the *Platform* dropdown if visible in the UI.
        - User selects a media from the *Media* dropdown if visible in the UI.
        - User enters the *Piece count* input field if visible in the UI.
        - User may enter a *description* (‘$Description’).
        - If data validation is successful for all the provided fields, then:
            - When the ***List item*** button is clicked:
                - If any field shown in the UI except ‘$Description’ is null, then:
                    - Display the **<u>Listing an Item</u>** form with error message.
                - Else:
                    - Add the new Item record.
                    - Display a success popup with the newly listed ItemID.
        - Else:
            - Display the **<u>Listing an Item</u>** form with error message.

View My Items

Task Decomp

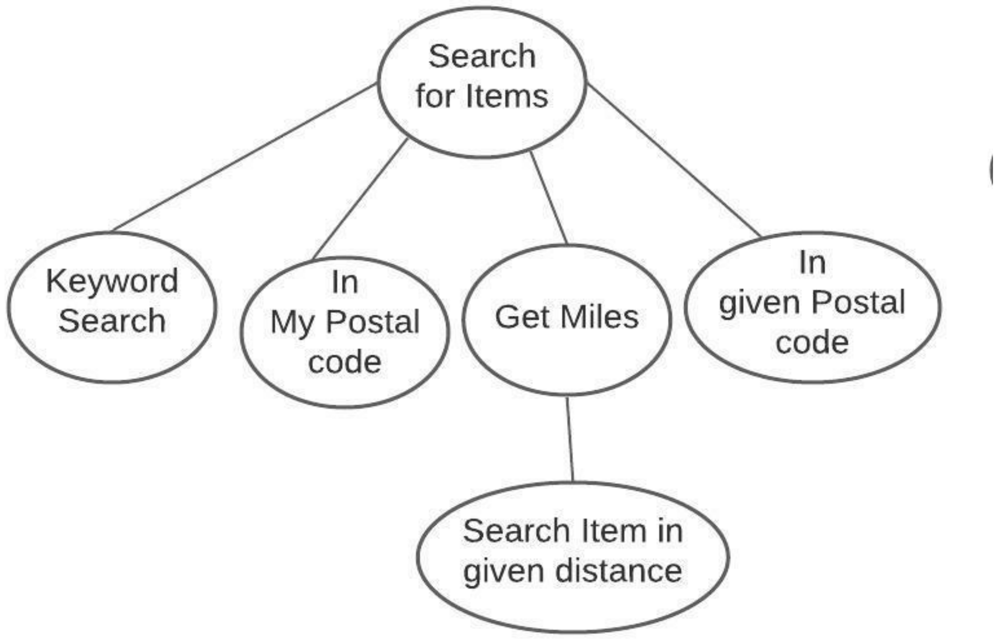
Lock Types: Read-only on Item table.
Number of Locks: Single.
Enabling Conditions: It is enabled when user click "My Items" in Main menu.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task is not needed. No decomposition needed. Order is not necessary.

Abstract Code

- User clicked on the ***My items*** button from the **Main Menu**.
- Run the **View My Items** task:
    - Find Items belonging to the user '$User' sorted by Item.ItemID (ascending order) .
        - Count the number of items for each game type and display them in a table in the UI together with the total item count.
        - Display a second table with the Item.ItemId, Item.Type, Item.Name, Item.Condition, Item.Description (if exists, truncated if more than 100 characters) and a ***Detail*** link for each of the items of user '$User' in ascending order of Item.ItemID.
- Upon click of the ***Detail*** link - Jump to the **View Item** task passing the Item.ItemID as a url parameter.

Search for Items

Task Decomp



Lock Types: Read-only on Item table if search item is based on keyword and Read-only on Item table and Location table if search item is based on My Postal code.
Number of Locks: 1 or 2 schema constructs are needed based on search selection.
Enabling Conditions: It is enabled when user click "Search Items" in Main menu.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task is needed.

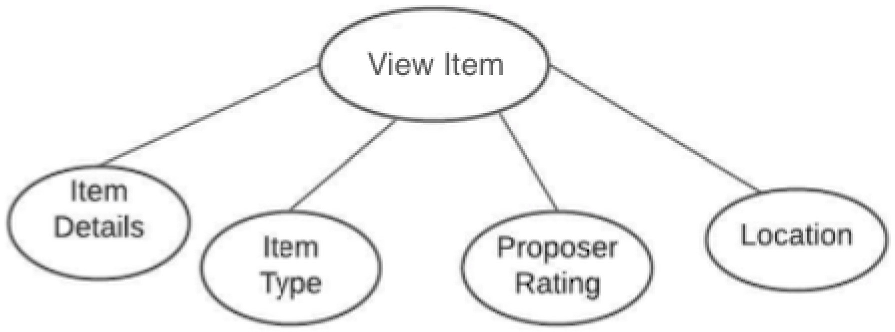
Abstract Code

- User clicked on the ***Search items*** button from the **Main Menu**.
- Run the **Search for item** task:
    - Find Locations and populate the *In postal code* ('$PostalCode') dropdown with a list of Location.PostalCode values.
    - User selects one of the 4 *search criteria* ('$SearchCriteria') radio buttons.
    - User enters the either the *By keyword* ('$Keyword') input field, *Within* X *miles of me* ('$DistanceWithin') input field, selects from the *In postal code* dropdown or does not enter anything depending on the *search criteria* radio button selection.
    - If data validation is successful for the relevant fields, then:

```
        ■ When the Search! button is clicked:
            ● If '$SearchCriteria' is "By keyword", then:
                ○ If '$Keyword' is null, then:
                    ■ Display the Searching for Items form with error message.
            ● Else if '$SearchCriteria' is "Within X miles of me", then:
                ○ If '$DistanceWithin' is NULL, then:
                    ■ Display the Searching for Items form with error message.
            ● Else if '$SearchCriteria' is "In postal code", then:
                ○ If '$PostalCode' is NULL, then:
                    ■ Display the Searching for Items form with error message.
            ● Find Items that fit the search criteria specified by the user
                ○ If the number of items that matched the search criteria is 0, then:
                    ■ Display the Searching for Items form with a "Sorry, no results found!" message.
                ○ Else:
                    ■ Display a table with the Item.ItemID, Item.Type, Item.Name, Item.Condition, Item.Description (if exists, truncated if more than 100 characters), the computed distance (rounded to tenths) from the user '$User' and a Detail link for each of the items that matched the search criteria sorted by ascending order of distance.
                        ● Upon click of the Detail link - Jump to the View Item task passing the Item.ItemID as a url parameter.
    ○ Else display the Searching for Items form with error message.
```

<u>View Item</u>

Task Decomp



Lock Types: Read-only on Item table and type-specific table if users view their own items. Read-only on Item table, type-specific table, Location table and swap table.
Number of Locks: 1 or more schema constructs are needed.
Enabling Conditions: It is enabled when user clicks "Details" in My Items or in Search Items.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task is needed.

Abstract Code

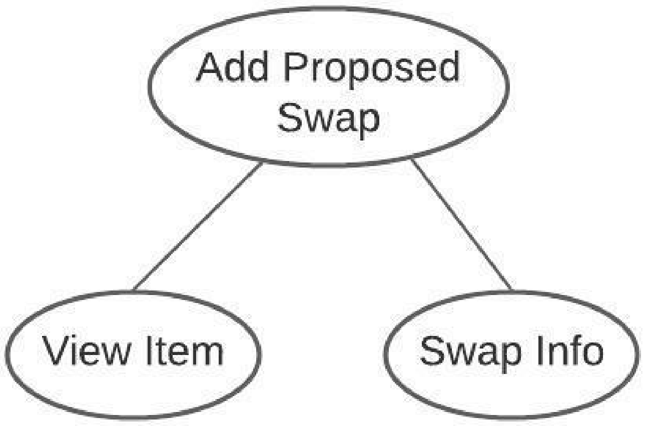
- Find Item where Item.ItemID is the ItemID provided in the url parameter.
  - Display the Item.ItemID, Item.Type, Item.Condition in the UI.
  - If Item.Platform is not null, then:
    - Display Item.Platform in the UI.
  - If Item.Media is not null, then:
    - Display Item.Media in the UI.
  - If Item.PieceCount is not null, then:
    - Display Item.PieceCount in the UI.
  - If the email of user that owns Item is not '$User', then:
    - Find User that owns Item.
      - Display User.Nickname in the UI.
    - Find the Location of user that owns Item.
      - Display Location.City, Location.State and Location.PostalCode in the UI.
      - Calculate the distance between the location of user '$User' and the location of user that owns Item.
        - If the distance > 0, then:

```
                        ■ Display the distance in the UI.
                        ■ If distance <= 25, then:
                            ● Highlight the distance in the UI in green.
                        ■ Else if distance <= 50, then:
                            ● Highlight the distance in the UI in yellow.
                        ■ Else if distance <= 100, then:
                            ● Highlight the distance in the UI in orange.
                        ■ Else:
                            ● Highlight the distance in the UI in red.
            ■ Find Swaps where the Swap.ProposerEmail is the owner of Item and
              Swap.ProposerRating is not null.
                ● Sum up each Swap.ProposerRating.
                ● Find Swaps where the Swap.CounterpartyEmail is the email of
                  the user that owns Item and Swap.CounterpartyRating is not
                  null.
                    ○ Sum up each Swap.CounterpartyRating.
                    ○ Add the Swap.ProposerRating sum with the
                      Swap.CounterpartyRating sum and divide by the Swaps
                      involving user that owns Item; Display the result as
                      *Rating* field in the UI.
            ■ If Item is eligible for swapping, then:
                ● Find Swaps where the Swap.ProposerEmail is '$User' and the
                  Swap.Status is 'COMPLETED' but Swap.CounterpartyRating is
                  null or find Swaps where the Swap.CounterpartyEmail is
                  '$User' and Swap.Status is 'COMPLETED' but
                  Swap.ProposerRating is null.
                    ○ Sum up the occurrences.
                    ○ If the number of unrated swaps <= 5, then:
                        ■ Find Swaps where the Swap.CounterpartyEmail
                          is '$User' and the Swap.Status is 'PENDING'.
                            ● Sum up the occurrences.
                            ● If the number of unaccepted swaps for
                              user '$User' <= 2, then:
                                ○ Display the **Propose swap** button in
                                  the UI.
                                    ■ Upon click **Propose swap**
                                      button - Jump to **Add
                                      Proposed Swap** task
```

passing the item.ItemID as a url parameter.

Add Proposed Swap

Task Decomp



Lock Types: Read-only on "details" link in view Item and Write on Swap table.
Number of Locks: Two.
Enabling Conditions: It is enabled when user has less than 2 unrated swap and clicks "Propose Item" from "Item Details".
Frequency: Same
Consistency (ACID): critical. Two or more users cannot request the same item.
Subtasks: Mother Task is needed.

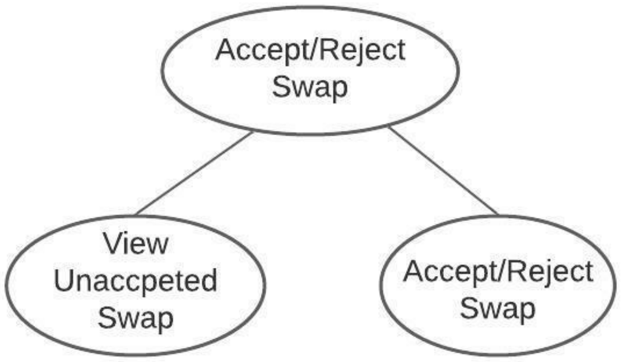
Abstract Code

- User clicked on the ***Propose swap*** button from **View Item**.
- Run the **Add Proposed Swap** task:
    - Find the Item where Item.ItemID is the ItemID passed as a url parameter.
        - Calculate the distance between the location of user '$User' and the location of user that owns Item.
            - If the calculated distance >= 100, then display a "The other user is <calculated distance> miles away!" warning message in the UI.
        - Display Item.name in the UI.
        - Find Items belonging to user '$User' that are eligible for swapping.

- Display a table with the Item.ItemID, Item.Type, Item.Name, Item.Condition and a *Select* radio button for each of the items of user '$User' eligible for swapping.
    - User selects an item through the *Select* radio button
        - Display the ***Confirm*** button in the UI.
            - Upon user clicks the ***Confirm*** button.
                - Add a new Swap record.
                - Display a success popup with a ***Return to Main Menu*** button.

Accept/Reject Swap

Task Decomp



Lock Types: Read on Swap table to list of unaccepted swap request and Read on Location table to find the distance and Write on Swap table. If swap accepted, Read on User table to get user's contact information.
Number of Locks: Two.
Enabling Conditions: It is enabled when user clicks "Unaccepted swap" in Main menu.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task is needed.

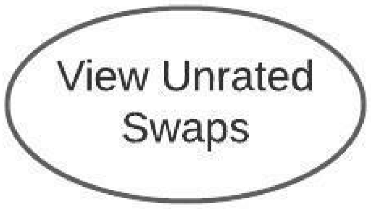
Abstract Code

- User clicked on the ***Unaccepted swaps*** link from **Main Menu**.
- Run the **Accept/Reject Swap** task:
    - Find Swaps where the Swap.CounterpartyEmail is '$User' and Swap.Status is 'PENDING'.

- Display a table with the Swap.ProposedDate, the ***name of desired item*** as a link, the name of the proposer, the rating of the proposer, the distance from the proposer, the name of the ***proposed item*** as a link, an ***Accept*** button and a ***Reject*** button for each Swap pending acknowledgement from user '$User'.
    - Upon:
        - Click ***name of desired item*** link.
            - Jump to **View Item** task passing the desired item's ItemID as a url parameter.
        - Click ***proposed item*** link:
            - Jump to **View Item** task passing the proposed item's ItemID as a url parameter.
        - Click ***Accept*** button.
            - Update the Swap record's status to 'COMPLETED' and AcknowledgeDate to the current date.
            - Find the User where User.Email is Swap.ProposerEmail.
                - Show a success pop up displaying the User.Email and User.FirstName.
                - If User.Share is True, then:
                    - Add User.PhoneNumber and User.PhoneType to the popup.
        - Click ***Reject*** button.
            - Update the Swap record's status to 'REJECTED' and AcknowledgeDate to the current date.

View Unrated Swaps

Task Decomp



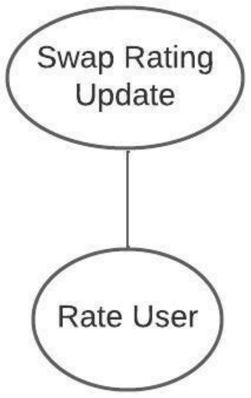
Lock Types: Read on Swap table to list of unrated swap request.
Number of Locks: Single.
Enabling Conditions: It is enabled when user clicks "UnRated swap" in Main menu.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task not is needed.

Abstract Code

- User clicked on the ***Unrated swaps*** link from **<u>Main Menu</u>**.
- Run the **View Unrated Swaps** task:
    - Find Swaps where the Swap.ProposerEmail is '$User' and the status is 'COMPLETED' but rating is null or find Swaps where the Swap.CounterpartyEmail is '$User' and the status is 'COMPLETED' but rating is null.
        - Display a table with the Swap.AcknowledgeDate, the role of the user '$User' in the Swap, the name of the proposed item, the name of the desired item, the nickname of the other user involved in Swap and a *Rating* ('$Rating') dropdown for each Swap pending a rating from user '$User'.
            - Upon select a rating from any of the *Rating* dropdowns:
                - If Swap.ProposerEmail is '$User', then:
                    - Jump to the **Swap Rating Update** task with the following parameters:
                        - '$SwapID' = Swap.SwapID
                        - '$IsCounterpartyRating' = True
                        - '$Rating'
                    - Jump to the **Swap Rating Update** task providing the SwapID,
                        - '$SwapID' = Swap.SwapID
                        - '$IsCounterpartyRating' = False
                        - '$Rating'

Swap Rating Update

Task Decomp



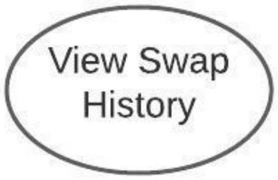
Lock Types: Write on Swap table.
Number of Locks: Single.
Enabling Conditions: It is enabled when user clicks "UnRated swap" in Main menu.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task is needed.

Abstract Code

- User selected a rating from the *Rating* dropdown from **Swap History, Rate Swaps or Swap Details**.
- If '$IsCounterpartyRating' is True, then:
    - Update the CounterpartyRating of the Swap record with Swap.SwapID == '$SwapID' to the value '$Rating'.
- Else
    - Update the ProposerRating of the Swap record with Swap.SwapID == '$SwapID' to the value '$Rating'.

View Swap History

Task Decomp



Lock Types: Read-only on User and Swap table.
Number of Locks: Single.
Enabling Conditions: It is enabled when user click "Swap History" in Main menu.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task is not needed. No decomposition needed.

Abstract Code

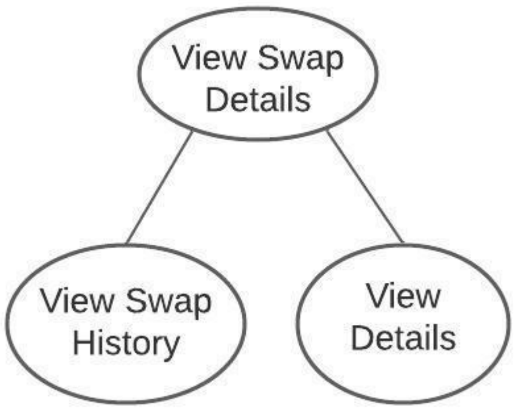
- User clicked on ***Swap History*** button from **Main Menu**:
- Run **Swap History** task: Query the swap table to get users swap information.
- Find the current User using '$User'
- Find Swaps where Swap.Status is either 'COMPLETED' or 'REJECTED' and either Swap.ProposerEmail is '$User' or Swap.CounterpartyEmail is '$User', sorted by Swap.AcknowledgeDate descending and Swap.ProposedDate ascending.
    - From the results compute the following statistics:
        - Count the number of Swaps proposed by user '$User' by checking Swap.ProposerEmail == '$User'
        - Count the number of accepted Swaps that were proposed by user '$User' by checking Swap.ProposerEmail == '$User' and Swap.Status == 'COMPLETED'
        - Count the number of rejected Swaps that were proposed by user '$User' by checking Swap.ProposerEmail == '$User' and Swap.Status == 'REJECTED'
        - Compute the % of rejected Swaps proposed by user '$User' by dividing the number of rejected Swaps that were proposed by user '$User' by the number of Swaps proposed by user '$User' and multiplying by 100%

    - Count the number of Swaps proposed to user '$User' by checking Swap.CounterpartyEmail == '$User'
    - Count the number of accepted Swaps that were proposed to user '$User' by checking Swap.CounterpartyEmail == '$User' and Swap.Status == 'COMPLETED'
    - Count the number of rejected Swaps that were proposed to user '$User' by checking Swap.CounterpartyEmail == '$User' and Swap.Status == 'REJECTED'
    - Compute the % of rejected Swaps proposed to user '$User' by dividing the number of rejected Swaps that were proposed to user '$User' by the number of Swaps proposed to user '$User' and multiplying by 100%
    - Display the swap statistics for user '$User' in a table in the UI.
  - For each Swap in the UI as a row in a table:
    - Display *Proposed Date* with Swap.ProposedDate
    - Display *Accepted/Rejected Date* with Swap.AcknowledgeDate
    - If Swap.Status is 'COMPLETED' , then:
      - Display *Swap status* with 'Accepted'
    - Else Display *Swap status* with 'Rejected'
    - If Swap.ProposerEmail is is '$User', then:
      - Display *My role* with 'Proposer'
      - If Swap.CounterpartyRating is not null, then:
        - Display *Rating* with Swap.CounterpartyRating
      - Else display a ('$Rating') *Rating* dropdown with values 1 to 5.
        - Upon select a rating from the *Rating* dropdown:
          - Jump to the **Swap Rating Update** task with the following parameters:
            - '$SwapID' = Swap.SwapID
            - '$IsCounterpartyRating' = True
            - '$Rating'
    - Else:
      - Display *My role* with 'Counterparty'
      - If Swap.ProposerRating is not null, then:
        - Display *Rating* with Swap.ProposerRating
      - Else display a ('$Rating') *Rating* dropdown with values 1 to 5.
        - Upon select a rating from the *Rating* dropdown:
          - Jump to the **Swap Rating Update** task with the following parameters:
            - '$SwapID' = Swap.SwapID

        - '$IsCounterpartyRating' = False
        - '$Rating'
    - Find proposed item in Item using Swap.ProposedItemId; Display *Proposed Item* with Item.Name
    - Find desired item in Item using Swap.CounterPartyItemId; Display *Desired Item* with Item.Name
    - Find other user in User using Swap.CounterPartyEmail; Display *Other User* with User.Nickname
    - Display ***Detail*** link
        - Upon click of the ***Detail*** link - Jump to the **View Swap Details** task passing the Swap.SwapID as a url parameter.

View Swap Details

Task Decomp



Lock Types: Read-only on User, Swap and Item table.
Number of Locks: Several schema constructs are needed.
Enabling Conditions: It is enabled when user click "Swap Details" in Swap History.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task is needed.

Abstract Code

- User clicked on ***Detail*** link from **Swap History**:
- Run **View Swap Details** task: Query the swap table to get users swap information:
    - Find Swap with the SwapID provided in the url parameter.

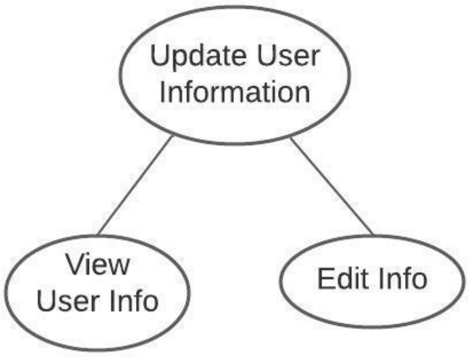
Table of contents

- Display the *Proposed* with Swap.ProposedDate, *Accepted/Rejected* with Swap.AcknowlegdedDate, *Status* with Swap.Status.
- If Swap.ProposerEmail is '$User' then:
    - Display *My role* as 'Proposer'
    - If Swap.CounterpartyRating is not null, then:
        - Display *Rating left* with Swap.CounterPartyRating
    - Else display a ('$Rating') *Rating* dropdown with values 1 to 5.
        - Upon select a rating from the *Rating* dropdown:
            - Jump to the **Swap Rating Update** task with the following parameters:
                - '$SwapID' = Swap.SwapID
                - '$IsCounterpartyRating' = True
                - '$Rating'
    - Find User using Swap.CounterpartyEmail
        - Display *Nickname* with counterparty User.Nickname, *Name* with User.FirstName, *email* with User.Email and *phone* with User.PhoneNumber
        - Find counterparty location in Location
            - Use Location.Latitude and Location.longitude to calculate distance to user '$User' and display as *Distance*.
- Else:
    - Display *My role* as 'Counterparty'
    - If Swap.ProposerRating is not null, then:
        - Display *Rating left* with Swap.ProposerRating
    - Else display a ('$Rating') *Rating* dropdown with values 1 to 5.
        - Upon select a rating from the *Rating* dropdown:
            - Jump to the **Swap Rating Update** task with the following parameters:
                - '$SwapID' = Swap.SwapID
                - '$IsCounterpartyRating' = False
                - '$Rating'
    - Find User using Swap.ProposerEmail
        - Display *Nickname* with proposer User.Nickname, *Name* with User.FirstName, *email* with User.Email and *phone* with User.PhoneNumber
        - Find proposer location in Location

            - Use Location.Latitude and Location.longitude to calculate distance to user '$User' and display as *Distance*.
        - Find Item using Swap.ProposerItemID
            - Display *Item* with Item.ItemID, *Title* with Item.Name, *Game Type* with Item.Type, *Condition* with Item.Condition
            - If Item.description is not null; Display *Description* with Item.Description
        - Find Item using Swap.CounterpartyItemId
            - Display *Item* with Item.ItemID, *Title* with Item.Name, *Game Type* with Item.Type, *Condition* with Item.Condition
            - If Item.Description is not null; Display *Description* with Item.Description.

Update User Information

Task Decomp



Lock Types: Write on User and Location table.
Number of Locks: Several schema constructs are needed.
Enabling Conditions: It is enabled when user click "Update my info" in Main menu.
Frequency: Same
Consistency (ACID): Not critical.
Subtasks: Mother Task is needed. Order is not necessary

Abstract Code

- User clicked on the ***Update my info*** button from **Main Menu**
- Run the **Update User Information** task:
    - Find Swap where the Swap.ProposerEmail is $User and the status is not 'COMPLETED'

        - If the occurrence is more than zero; Display a popup with an error message.
    - Find Swap where the Swap.CounterpartyEmail is $User and the status is not 'COMPLETED'
        - If the occurrences is more than zero;Display a popup with an error message
    - Find Swap where the Swap.ProposerEmail is $User and the status is 'COMPLETED' but rating is null
        - If the occurrences is more than zero;Display a popup with an error message
    - Find Swap where the Swap.CounterpartyEmail is $User and the status is 'COMPLETED' but rating is null
        - If the occurrences is more than zero;Display a popup with an error message
    - Find the current User using $User; Display the user User.Email in the uneditable *Email* input field, User.Nickname in the *Nickname* ('$Nickname') input filed, User.FirstName in the *First Name* ('$FirstName') input field, User.LastName in the *Last Name* ('$LastName') input field, User.Password in the *Password* ('$Password') input field, User.PhoneNumber in the *Phone Number* ('$PhoneNumber') input field (if available), check the *Show phone number in swaps* ('$ShowPhoneNumber") check box if User.Share is True and User.PhoneType in *Type* ('$PhoneNumberType') dropdown (if available)
    - Find the Location of user $User
        - Display Location.City in the *City* ('$City') input field, Location.State in the *State* ('$State') input field and Location.PostalCode in the *Postal Code* ('$PostalCode') dropdown
    - Mask the '$Password'
    - Find the current User location in Location; Display Location.City in the *City* ('$City') input field, Location.State in the *State* ('$State') input field and Location.PostalCode in the *Postal Code* ('$PostalCode') dropdown
    - User updates some or all of the fields
        - When the ***Update*** button is clicked
            - Find User where User.Email is not $User and User.PhoneNumber == '$PhoneNumber'
                - If the occurrences is more than zero; Display a popup with an error message
            - Update the user record for user $User with '$FirstName', '$LastName', '$Nickname', '$Password', '$PhoneNumber', '$City', '$State' and '$PostalCode'