Table of Contents

Data Type

User

Attribute	**Data Type**	**Nullable**
Email	String	Not Null
Password	String	Not Null
First_name	String	Not Null
Last_name	String	Not Null
Nick_name	String	Null
Phone_number	String	Null
Phone_type	String	Null
Phone_shareable	String	Null

Note: Nick_name, and Phone_ attributes are nullable since they may be missing / not applicable for some users, e.g. for users who did not provide a nickname or number.

Address

Attribute	**Data Type**	**Nullable**
Postal_code	String	Not Null
City	String	Not Null
State	String	Not Null
Latitude	Numeric	Not Null
Longitude	Numeric	Not Null

Item

Attribute	**Data Type**	**Nullable**
Title	String	Not Null
Game_type	String	Not Null
Condition	String	Not Null
Description	String	Not Null

Platform	String	Nullable
Media	String	Null
Piece_count	String	Null
Swappable (derived)	Bool	Not Null

Note: Platform, Media, and Piece_count are nullable since they may be missing / not applicable for some items, e.g. for non-VideoGame or non-Puzzle items.

Swaps

Attribute	**Data Type**	**Nullable**
Propose_date	Date	Not Null
Accept_reject_date	Date	Null
Status	String	Not Null
Proposer_rating	String	Null
Counterparty_rating	String	Null

Note: Proposer_rating and Counterparty_rating are nullable because they may be missing / not applicable for some swaps, e.g. before the swap has been rated

Business Logic Constraints

User

- Users who are new to GameSwap must register first
- Users who have an existing account will not be able to register with the same email
- A registered user cannot update the email
- A phone number can only be registered to a single user
- A user must rate each other after a successful swap on a scale of 0-5
- A registered user cannot update items if the user has any unapproved swaps or unrated swaps
- A user can only select post code from a given list
- Only one user can use the system at a time

Swap

- A user cannot swap if the user have no items listed
- Items have a pending swap (not yet accepted or rejected) are not able for swapping
- The user cannot swap items with themself
- If a swap is rejected, the item-for-item swap cannot be proposed again
- An item that have been swapped cannot be swapped again unless relisting with updated condition or description
- If users have more than 2 unrated swaps or more than 5 unaccepted swaps, they cannot list a new item
- The Accept/Reject Date and status are null before the conterparty making decision
- The Proposer Rating and Counterparty Rating are null before the user rating for the other
- Possible status are pending, rejected and accepted

Item

- Only five conditions are available (Mint, Like New, Lightly Used, Heavily Used, Missing parts/Damaged)
- Only the first 100 characters of the description are displayed in the “my items” form
- User can only search by one category at a time (keyword/ in my postal code/ within X miles/ in postal code X)

Task Decomposition

Formatting Key for Abstract Code:
Bold Underlined – Form Example: **Main Menu** form
Bold Italics* –** Buttons Example: ***Save button
Bold – Task Example: **Login** task

Login

Task Decomp

Lock Types: read USER
Number of Locks: 1

- Read USER entity 1x

Enabling Conditions: None
Frequency: once per login
Consistency (ACID): Order is not critical
Subtasks: One Task

Abstract Code

- Render *Email/phone number* text input field,
    - When value is changed, set variable $username to current value
- Render *Password* text input field
    - When value is changed, set variable $password to current value
- Register ***Login*** button
- Render ***Register*** button
- When ***Register*** button is clicked, go to **Register** task
- When ***Login*** button is clicked:
    - IF USER record with identifier matching $username is found
        - IF found USER'S password is equal to $password
            - Login USER, go to the **View Main Menu** task
        - ELSE (USER's *password* != $Password')
            - Render error message
    - ELSE: email and password not registered in the database or invalid inputs
        - Render error message

Register

Task Decomp

Lock Types: write USER
Number of Locks: 1

- Write USER entity 1x to create new user
- Note: No lock needed on ADDRESS since the entity is immutable

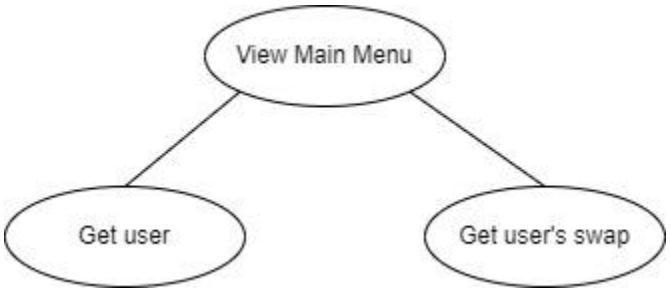
(lock not needed on Address, since it cannot be updated)
Enabling Conditions: User clicks ***Register*** button from **Login** task
Frequency: once per registration
Consistency (ACID): Order is not critical
Subtasks:

- One task can perform registration

Abstract Code

- User enters required text input fields *email*, *password*, *firstname*, *lastname*, *nickname* and optional fields *phone_number, phone_type, show_phone_in_swaps* in registration form
- User selects postal code and the system fills in a matching City and State in the user profile from the ADDRESS entity
- When current user hits ***Register*** button
    - Try to insert new USER with values from text input fields
        - If successful
            - Go to **View Main Menu** task with “welcome” message on top of the menu that includes the user’s first and last name
        - Else
            - Show error message from inserting user, e.g. “E-mail already in use” and/or “Phone number already in use”

View Main Menu



Task Decomp

Lock Types: read USER, read ITEMS, read SWAP
Number of Locks: 3

1. Read USER entity 1x to get user's name and items
2. Read ITEM entity 1x to get swaps associated with user's items
3. Read SWAP entity 1x to get user's swaps

Enabling Conditions: Successful login, user registration, swap proposal confirmed, all swaps accepted or rejected
Frequency: Medium - All 3 have the same frequency, all are executed multiple times per day
Consistency (ACID): Not required, read only. Only one user will use the system at a time.
Subtasks:

- Get User
- Get User's Swaps

Abstract Code

- Subtask Get USER:
    - find one USER where user email matches the user email from the page URL / HTTP request / session
    - Save result to variable $user
- Subtask Get User's Swaps:
    - find all ITEMs where user is owner and all SWAPs where user's ITEMs are involved
    - Save result to variable $users_swaps
- Render **Main Menu**
    - Render Welcome Header: "Welcome {$user.firstname} {$user.lastname}
    - Render My Rating Panel:
        - IF count of $users_swaps is less than one, replaced calculated rating with string "None"
        - ELSE calculated average rating by accumulating all of the ratings of the user from the user's completed swaps and dividing by the number of completed swaps and save to variable $rating. Display "My Rating: { $rating }

- Render Unaccepted Swaps Panel: return “Unaccepted Swaps: {count of swaps where user is counterparty and $swap.status is not accepted}
    - **IF** count of unaccepted swaps is greater than 0, wrap count of unaccepted swaps in hyperlink with URL to **Accept/Reject Swaps** form
    - **IF** the count of unaccepted swaps where current date minus $proposal_date is greater than 5 days is greater than 0, OR count of unaccepted swaps is greater than 5, wrap hyperlink in styling bold and red
- Render Unrated Swaps Panel: return “Unrated Swaps: {count of swaps where (user is $counterparty and $counterparty_rating is null OR user is $proposer and $proposer_rating is null}”
    - IF count of unrated swaps is greater than zero, wrap count of unrated swaps in hyperlink with URL to **Rate Swaps** task
    - IF count of unrated swaps is greater than zero, wrap hyperlink in styling bold and red
- Render ***List Items*** button: display hyperlink to **List Item** task page
- Render ***My Items*** button: display hyperlink to **View Item Details** task page
- Render ***Search Items*** button: display hyperlink to **Search Items** task page
- Render ***Swap History*** button: display hyperlink to **View Swap History** task page
- Render ***Update My Info*** button: display hyperlink to **Update My Info** task page
- Render ***Logout*** button: on click, invalidate user session, redirect to **Login** task page

Update User Info

Task Decomp

Lock Types: read ITEM, read SWAP, read USER, write USER
Number of Locks: 4

1. Read **ITEM** entity 1x to find user's swaps
2. Read **SWAP** entity 1x to find unresolved swaps
3. Read **USER** entity 1x to find user's data
4. Write **USER** entity 1x to update
5. Note: no locks needed on ADDRESS since the entity is immutable

Enabling Conditions: Logged in user clicks ***Update My Info*** button from **Main Menu**
Frequency: Low, when user phone or address changes, when user wants to change password, when user wants to share / unshare phone.
Consistency (ACID): Not required. A user will only ever update their own info, therefore there is not a high frequency or level of contention for the document under normal circumstances. Constraints in the business logic prevent a user from updating their info if they have pending swaps, therefore isolation will not need to be enforced at the database level.

Subtasks:

- Check and render unapproved or unrated swaps exception
- Get User Data
- Update User

Abstract Code:

- Subtask: Check and render unapproved or unrated swaps exception
    - find SWAPs where the current user is $counterparty or $proposer and $swap_status is not accepted OR where $status is accepted and the user has not rated the other participant. Save result to variable $unapproved_or_unrated_swaps
    - IF length of $unapproved_or_unrated_swaps is greater than 0:
        - display "You have {length of $unapproved_or_unrated_swaps} unapproved or unrated swaps. Please resolve pending swaps before updating user information."
    - ELSE continue to Get User Data subtask and render **Update User Information** form
- Subtask: Get User Data
    - Get data for current user and save to variable $user_data
    - Get address for current user and save to variable $user_address

- Render **<u>Update User Information</u>** form
    - Define form template **<u>Update User Information</u>**:
        - Email**,** Text Input:
            - Label is “Email”
            - Disabled is True
            - Starting value is $user_data.email
        - *Nickname***,** Text Input:
            - Label is “Nickname”
            - Starting value is $user_data.nickname
        - *Password***,** Text Input:
            - Label is “Password”
            - Display is “********”
            - Starting value is null
        - *City***,** Text Input:
            - Label is “City”
            - Starting value is $user_address.city
        - *First Name***,** Text Input:
            - Label is “First Name”
            - Starting value is $user_data.first_name
        - *State***,** Text Input:
            - Label is “State”
            - Starting value is $user_address.state
        - *Last Name***,** Text Input:
            - Label is “Last Name”
            - Starting value is $user_data.last_name
        - *Postal Code***,** Text Input:
            - Label is “Postal Code”
            - Starting value is $user_address.postal_code
        - *Phone Number***,** Text Input**:**
            - Label is “Phone number (optional)
            - Starting value is $.phone_number else “”
        - *Type***,** Dropdown List:
            - Label is “Type”
            - Starting value is $user_data.phone_type
        - *Shown Phone Number***,** Check Box:
            - Starting value is $user_data.phone_shareable
        - ***Update*** button: Text is “Update”, IF clicked, execute Update User subtask
    - Render **<u>Update User Information</u>** form with values from $user_data and $user_address
- Subtask Update User:
    - Define variable $update_user_payload
    - For each field in **<u>Update User Information</u>** form where value is not null and value is not equal to $user_data.{value_name} append field name and value as key, value to $unapproved_or_unrated_swaps

- IF $update_user_payload is not empty, update current user's USER entity with $update_user_payload
    - IF error on updating, display friendly error message:
        - Format and display error to user, e.g. **"**The phone number you provided {update_user_form.phone_number} is already in use. Please use another phone number."
    - ELSE return to **View Main Menu** task

List New Item

Task Decomp

Lock Types: write ITEM, read SWAP, read ITEM
Number of Locks: 3

1. Read ITEM 1x to associate swaps with the current user
2. Read SWAP entity x to determine user's ability to list a new item
3. Write ITEM entity 1x to list a new item

Enabling Conditions: user logged in, user clicks ***List Item*** button
Frequency: Low, less than 50 insertion a day
Consistency (ACID): consistency is not important in this case.
Subtasks:
Mother Task is not needed. No decomposition needed

Abstract Code:

- Show *Game type* drop down list, *Title* text input field, *Condition* text input field and *Description* text input field. The contents of each field will be accessible via a variable name, such as $game_type.
- If $game_type is selected as "Jigsaw Puzzle", show *Piece count* field
- If $game_type is selected as "Video Game", show *Platform* and *Media* fields
- If $game_type is selected as "Computer Game", show *Platform* field
- If user clicks ***List Item:***
    - Verify that the user has less than 3 unrated swaps, and less than 6 unaccepted swaps. Prompt error if the verification does not pass
    - Try to insert data from the form into ITEM table
        - IF success:
            - A success message will be displayed to the user with the item number (generated on insert)
        - IF error:
            - A friendly error message will show the user any issues from the request

Search Items

Task Decomp

Lock Types: Read ITEM, Read USER, Read SWAP
Number of Locks: 3

- Read ITEM entity 1x to pull up item info
- Read SWAP entity 1x to get swap eligibility
- Read USER entity 1x to identify address info
- Note: no locks needed on ADDRESS since the entity is immutable

Enabling Conditions:

- User is logged in
- User clicks ***Search*** button

Frequency: This task can be repeated as many times as a user wants, with the same or differing inputs.
Consistency (ACID): No consistency needed. The assumption is that only one user is accessing the system at a time, so there is no danger of a concurrent update. Furthermore, since this search does not do any writes, there is no need for read-your-writes consistency.

Subtasks: All of the tasks require a read lock on all 4 tables in order to display all of the information (USER table, SWAPS table, ADDRESS table, ITEM table). Although only one subtask will be executed, hence no ordering, the mother task is required to decide/orchestrate which subtask to execute.
Mother Task: Search Item

- Search by keyword OR
- Search by “in postal code” OR
- Search by “in my postal code” OR
- Search by “within X miles of me”

Abstract Code

- Render **Search for Item** form with appropriate input fields for *By Keyword, In My Postal Code, Within X Miles,* and *In Postal Code* searches. The contents of each text input field will be accessible via a variable such as $keyword for the *By Keyword* field.
- Do nothing until user clicks ***Search***
- When the user clicks ***Search***:
    - If the user filled in the *By Keyword* text field:
        - Find all swap-eligible items matching ‘$keyword’ with item’s ‘item.name’ or strings in ‘item.description’
        - For each item:

        - Find the user associated with the item and the location associated with the user
    - If the result is empty:
        - display “Sorry, no results found”. Return the user to the **Search for Item** form.
    - Sort the list of items by distance, then item id, ascending.
    - For each item, display item ID, Game Type, Title, Condition, Description, Distance and a link called ***Details***, with a blue highlight around the column that contains a matching key word
        - If ‘item.description’ has more than 100 characters, display (...) at the end of first 100 characters
- Else If the user checked the ***In My Zip Code*** button:
    - Find the user’s location
    - Find all swap-eligible items with a user with the same zip code
    - If the result is empty,
        - display “Sorry, no results found”. Return the user to the **Search for Item** form.
    - Sort the list of items item id, ascending.
    - For each item, display item ID, Game Type, Title, Condition, Description, Distance and a link called ***Details***
        - ***If ‘item.description’ has more than 100 characters, display (...) at the end of first 100 characters***
- Else If the user selected a value in the ***Within X Miles*** dropdown:
    - Find the user’s location
    - Find all swap-eligible items with a user with a distance less than X miles
    - For each item:
        - Find the user associated with the item and the location associated with the user
    - Sort the list of items by distance, then item id, ascending.
    - Display item ID, Game Type, Title, Condition, Description, Distance and a link called ***Details***
        - If ‘item.description’ has more than 100 characters, display (...) at the end of first 100 characters
- Else if the user entered a postal code:
    - If the postal code is invalid:
        - Display an error message to the user. Return to the **Search for Item** form
    - Find all swap-eligible items with a user with the same zip code
    - If the result is empty,
        - display “Sorry, no results found”. Return the user to the **Search for Item** form.
    - Sort the list of items item id, ascending.
    - For each item, display item ID, Game Type, Title, Condition, Description, Distance and a link called ***Details***

        - If ‘item.description’ has more than 100 characters, display (...) at the end of first 100 characters
- Then, do nothing until the user clicks something
- When the user clicks ***Details*** on a specific item, run the **View Items Details** task for that item.

View Item Details

Task Decomp

Lock Types: Read ITEM, USER and SWAP (to calculate rating) entities
Number of Locks: 3

1. Read ITEM entity 1x to get item details
2. Read USER entity 1x to get owner details
3. Read SWAP entity 1x to get rating information
4. Note: no locks needed on ADDRESS since the entity is immutable

Enabling Conditions:

- Current user is logged in.
- User clicked on the ***Details*** button in the Search Items results screen
- User clicked on the ***Details*** button in the My Items results screen

Frequency: Frequently, expecting each logged-in user to view a few items. Around 300 views a day.
Consistency (ACID): Consistency is not critical. Subtasks need to be done in sequence.
Subtasks:

- Get item's info
- Get owner's info (including calculating rating and calculating distance)
- Verify if the current user can propose a new swap

Abstract Code

When the user clicks on **Details:**

- Display all the item's attributes: item ID, title/name, type-specific attribute, and description (if applicable)
- IF the item belongs to another user:
    - In addition to the above display, display the item's owner information {user.$nickname}, {user.$city}, {user.$state}, {user.$postalcode}
    - Owner's rating need to be calculated based on the records in the swap table (method is also mentioned in **view main menu**)
    - Calculate the distance between the proposer and the counterparty.
    - IF the distance is:
        - between 0.0 and 25.0 miles:
            - Display distance with highlighted green background
        - Between 25.0 and 50.0 miles:
            - Display distance with highlighted yellow background
        - Between 50.0 and 100.0 miles

        - Display distance with highlighted orange background
    - Over 100.0 miles:
        - Display distance with highlighted red background
- IF the current user does not have more than 2 unrated swaps or more than 5 unaccepted swaps and the item is available for swapping:
    - IF the user clicks the ***Propose Swap*** button, go to **Proposing a Swap** form

View “My Items” Section

Task Decomp

Lock Types: read ITEM, read SWAP
Number of Locks: 2

- Read ITEM entity 1x for a list of all of current user’s items
- Read SWAP entity 1x to determine item’s status (pending swap, already swapped, etc)

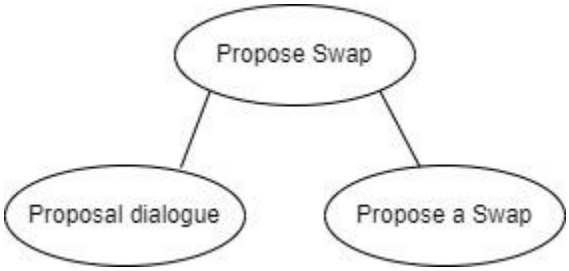
Enabling Conditions: Current user is logged in
Frequency: Everything within the task gets done once per task.
Consistency (ACID): No consistency needs
Subtasks: No need to break this into subtasks.

Abstract Code

- Define variable $user_email as user email from URL / HTTP request
- Find ITEMs where $user_email is owner and item is not associated with a pending or complete swap. Save results to variable $user_items
- Create a table labeled “Item Counts”. For each game type, create a column header named after the type.
- Aggregate the count of items within $user_items that match the game type. Render this in the Item Counts table.
- Under the label “Item Counts”, display a table called “My Items” with a column for Item Number, Game Type, Title, Condition, Description, and an untitled column to use for links to the item’s details.
- For each item in $user_items. Render a row in the table with the appropriate attribute of the item in each column and a ***Details*** link in the last column.
- Do nothing
- If the ***Details*** link is clicked for an item
    - Do the **View Details** task for that item.

Propose Swap

Task Decomp



Lock Types: read USER,, read ITEM, read SWAP, write SWAP
Number of Locks: 4

1. Read USER entity 2x to identify proposer and counterparty address
2. Read ITEM entity 1x for a list of current user's items
3. Read SWAP entity 1x to filter the items already in a swap
4. Write SWAP entity 1x to insert a new swap in pending state
5. Note: no locks needed on ADDRESS since the entity is immutable

Enabling Conditions: Current user is logged in and has less than three unrated swaps and less than six unaccepted swaps in order to see "***Propose Swap***" button. Neither item was previously swapped. User has clicked ***Propose Swap*** from the **View Details** page.
Frequency: Upon entering the task, the user will only do each read subtask once. The write subtask is optionally performed.
Consistency (ACID): No consistency requirements since only one user will be logged in to the system at a time.
Subtasks:

1. **Proposal dialogue**
    a. Read USER **and** ADDRESS entities to calculate distance
    b. Read ITEM to show current user's swap-eligible items
2. **Propose a Swap**
    a. Write SWAP to enter a new pending swap

Abstract Code

Subtask 1: Proposal Dialogue

- Calculate distance from current/proposer USER to the counterparty USER by identifying each ADDRESS via the USER
    - IF the distance between current/proposer ADDRESS and counterparty ADDRESS is greater than or equal to 100 miles, display a warning with distance to counterparty in miles
- Define variable $eligible_items
- Find ITEMS where owner is the current USER and item is not associated with a pending or complete SWAP and save the list of result items to the $eligible_items variable
- Display a table with column headers "Item #", "Game Type", "Title", and "Condition". An additional empty column will be shown.

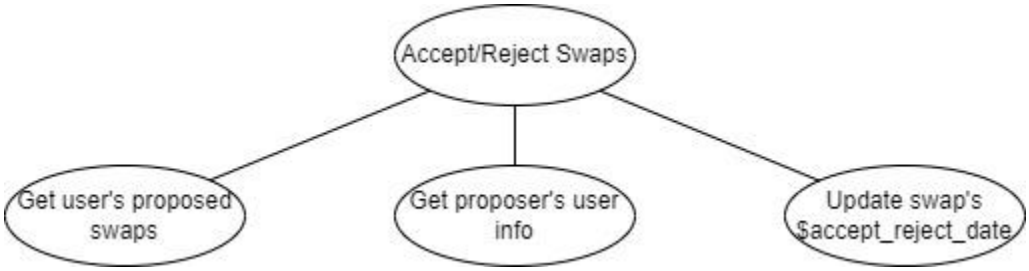
- For each item in $eligible_items, create a row in the table and populate the item ID, game type, title, and condition in the matching column. In the final column with the empty header, display a ticker box labeled “Select”
- Do nothing.
- When a user clicks on one of their item’s “Select” box, record that item’s ID in the active session as the proposed item, and create a button ***Confirm***.
- Do nothing.
- WHEN the user clicks on another item
    - Record that item’s ID in the active session as the proposed item
    - Deselect all other tickers associated with other items.
- WHEN the user clicks ***Confirm***
    - Execute **Propose a Swap** subtask with the current session’s proposed item ID

Subtask 2: **Propose a Swap**

- Create an instance of the SWAP entity representing the two items involved in the swap.
- Display confirmation message with a hyperlink button to the **View Main menu** task

Accept/Reject Swaps

Task Decomp



Lock Types: read SWAP, write SWAP, read ITEM, read USER
Number of Locks: 4

1. Read SWAP entity 1x to find user's proposed swaps
2. Read ITEM entity 1x to find item title
3. Write SWAP entity 1x to update accept_reject_date in SWAP entity/table
4. Read USER entity 1x to find proposer's info if user clicks ***Accept*** on a proposed swap

Enabling Conditions: Current user is logged in, a SWAP is proposed and current user is counterparty, user clicked on the ***Unaccepted swaps*** button in the **Main Menu** form
Frequency: One acceptance/rejection typically occurs per proposed Swap
Consistency (ACID): Tasks need to be done in sequence. A mother task is required.
Subtasks:

- Get current user's proposed swaps
- Get the proposer's user info
- Update swap $accept_reject_date

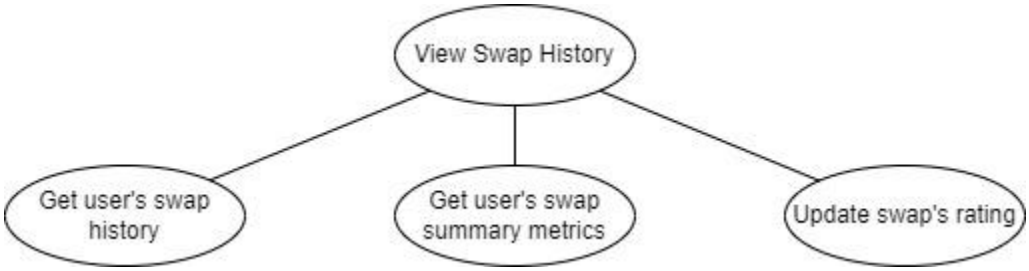
Abstract Code

- Define variable $user_email as user email from URL / HTTP request
- Find SWAPs where counterparty ITEM is associated with a USER which matches the $user_email and status is not ACCEPTED or REJECTED, and save to variable $proposed_swaps
- Sort $proposed_swaps by "date" proposed (in ascending order)
- Render a table with headers "Date", "Desired Item", "Proposer", "Rating", "Distance", "Proposed Item", and a final empty column
- For each swap in $proposed_swaps**,** render a row and populate each column with the appropriate value. Round number rating to hundredths. Round Distance to tenths. In empty column show ***Accept*** and ***Reject*** buttons
- Do nothing until the user clicks either ***Accept*** or ***Reject***.
- If the user clicks ***Accept*** on a proposed swap
    - IF phone information is present and phone is shared
        - Display dialog with proposer's $email, $first_name, $phone_number/type (if available) and sharing option (if set).
    - IF phone number is available, but not shareable
        - display as not available
    - Record a value of "ACCEPTED" for the status of the swap in the SWAP table

        - Record a value of current time in UTC for the $accept_reject_date of the swap in the SWAP table
        - Jump to **Rate Swaps** task
- Else If the user clicks ***Reject*** on a proposed swap
    - Record a value of “REJECTED” for the status of the swap in the SWAP table
    - Record a value of current time in UTC for the accept_reject_date of the swap in the SWAP entity
- If the number of proposed swaps is zero
    - User will be returned to **Main Menu**

View Swap History

Task Decomp



Lock Types: read SWAP, read ITEM, read USER, write SWAP
Number of Locks: 4

1. Read SWAP entity 2x to calculate user's summary metrics and to find user's swap history
2. Read ITEM entity 1x to get ITEMs' title
3. Read USER entity 1x to get USER'S role and name
4. Write SWAP entity 1x to add rating

Enabling Conditions: Current user is logged in
Frequency: Medium - The subtasks have the same frequency, all are executed multiple times per day
Consistency (ACID): need to display information first and update rating when the user chooses a rating.**Subtasks**:

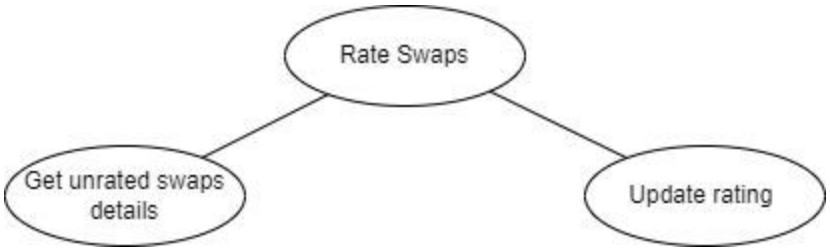
- Get user's swap history
- Get user's swap summary metrics
- Update a SWAP's rating

Abstract Code

- Calculate Rejected % for proposer USER and counterparty USER by dividing the rejected swaps by total swaps
- Render summary metrics: summary, logged in user's total swaps proposed, total received, subtotals for accepted and rejected, and % rejected
    - If % rejected (rounded to tenths) >=50.0%
        - highlight % rejected in red
    - Else
        - Do nothing
- For each swap for current user, render a row with acceptance/rejection date, swap proposed date, proposed item title, desired item title, other user's nickname, rating
    - If swap has not been rated
        - Render dropdown list with rating options as integers 0 through 5
        - When rating is selected update the associated SWAP entity's rating and refresh the page
    - If User clicks ***Detail*** button
        - Jump to **View Swap Details** task

Rate Swaps

Task Decomp



Lock Types: read SWAP, write SWAP**,** read ITEM, read USER
Number of Locks: 4

1. Read SWAP entity 1x to find all details of a selected swap
2. Write SWAP entity 1x to update proposer_rating and counterparty_rating
3. Read ITEM entity 1x to find all details of associated items
4. Read USER entity 1x to find all details of associated users

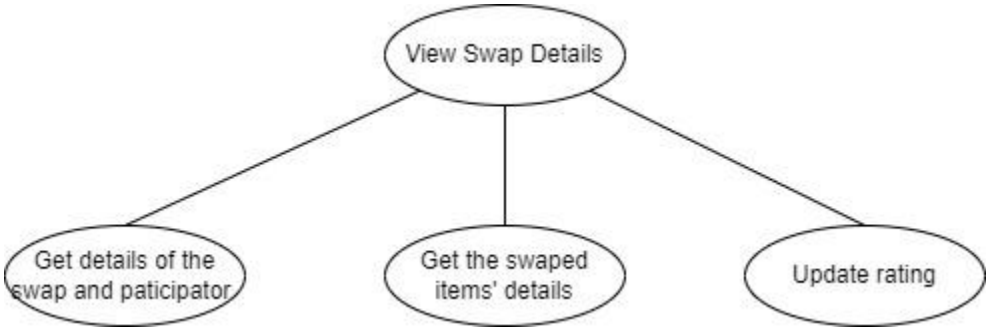
Enabling Conditions: Current user is logged in, user is the proposer or counterparty of swap, User clicked on the ***Unrated Swaps*** panel in the **View Main Menu**
Frequency: Low, one rating for the proposer and one rating for the counterparty typically occurs per proposed Swap
Consistency (ACID): Consistency is not critical. Can only update ratings for the displayed unrated swaps.
Subtasks:

- Get user's unrated swap details
- Update Swap rating

Abstract Code

- User clicked on the ***Unrated Swaps*** panel in the **Main Menu**
- Display all accepted swaps for user, where user has not rated
    - Populate a table with the swap acceptance date, the user's role in the proposal (proposer or counterparty), proposed item title, desired item title, other user's nickname, and a drop-down rating list (0 – 5), ordered by acceptance date descending
        - If current user chooses a rating for an unrated swap
            - Record the rating in the database
            - IF more swaps are unrated
                - refresh the **Rate Swaps** form
            - IF no additional swaps need rating
                - return the user to the **Main Menu**

View Swap Details



Task Decomp

Lock Types: read SWAP, write SWAP, read USER, read ITEM
Number of Locks: 4

1. Read SWAP entity 1x to find all details of a selected swap
2. Read USER entity 1x to find all details of proposer
3. Write SWAP entity 1x to update proposer_rating, counterparty_rating
4. Read ITEM entity 1x to find all details of items

Enabling Conditions:

- Current user is logged in
- User is the proposer or counterparty of swap
- User clicked on the ***Detail*** button in the **Swap History** form

Frequency: Medium - The two read subtasks have the same frequency. Each logged-in user may view multiple times. Reads will occur approximately 300 times a day. The write subtask may optionally be called.
Consistency (ACID): Consistency is not critical. Order does not matter.
Subtasks:

- Get specific SWAP details and the associated ITEM details
- Get swap's proposer details from USER entity
- Update Swap rating (only rendered if swap does not have rating already)

Abstract Code

- User clicked on the ***Detail*** button in the **Swap History** form
- Render Swap Detail Page
    - Display the selected swap details (proposed date, accepted/rejected date, swap status, swap type) and user rating, other user's nickname (if applicable), distance
    - IF selected swap is an accepted swap
        - Display contact details (first name, email, and phone number/type if available and sharing option is set from the USER entity), and details for both items (all available attributes in the ITEM entity)
        - IF selected swap is not rated

- Allow a rating to be submitted with rating dropdown (0-5)
- IF user submits rating
    - refresh/update the form after submission