Table of Contents:

Login

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:

      ```
      SELECT password from User where email = '$Email' or
      phoneNumber = '$Password'
      ```

    - If the User record is found and User.Password != '$Password':
        - Go back to the **Login** form, with an error message.
    - Else:
        - Store login information as session variable '$User'.
        - Go to the **Main Menu** page.
- Else *Email/phone number* and *Password* input fields are invalid, display the **Login** form with error message.

Register

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.

      ```
      SELECT postal_code  from Location
      ```

    - 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.

```
SELECT city, state FROM Location WHERE postal_code = '$PostalCode'
```

- User optionally provides phone number information:
    - User enters the 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.

```
INSERT INTO PhoneNumber('phone_number', 'type', 'share_phone_number')
VALUES ('$PhoneNumber', '$Type', 'ShowPhoneNumber')
```

- 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'.

```
INSERT INTO User('email', 'postal_code', 'first_name', 'last_name',
'nickname', 'password') VALUES ('$Email', '$PostalCode', '$FirstName',
'LastName', '$NickName', '$Password' )
```

            - Go to the **Main Menu**.
- Else Display the **User Registration** form with error message.

View Summary

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.

```
SELECT first_name, last_name FROM User where email = '$Email'
```

    - Join tables RatedSwap and Swap on swap_id and email. Use the aggregate function AVG to calculate the rating as MyRating

```
SELECT IFNULL(AVG(rating),'0') as MyRating
FROM RatedSwap r INNER JOIN
        Swap s
ON r.email = '$Email'' AND
r.swap_id = s.swap_id AND
(r.email = s.proposer_email OR r.email = s.counterparty_email)
```

    - Find out the count of unaccepted swaps from Swap where $Email = proposer_email AND swap_id NOT present in AcknowledgedSwap

```
SELECT COUNT(1) unaccepted_swap_count
FROM Swap s
WHERE s.proposer_email = '$Email'' and
s.swap_id not in (select swap_id from AcknowledgedSwap)
```

    - Find out the count of unrated swaps from Swap where $Email = proposer_email or counterpart_email AND swap_id might be present as ACCEPTED/REJECTED in AcknowledgedSwap but NOT present in RatedSwap

```
((SELECT COUNT(1) unrated_swap_count from Swap s
JOIN AcknowledgedSwap a
ON s.swap_id = a.swap_id
WHERE a.status = 'ACCEPTED'
AND (s.proposer_email = '$Email' OR s.counterparty_email = '$Email')
AND a.swap_id NOT IN
(SELECT swap_id from Ratedswap where email = '$Email')));
```

- 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

Abstract Code

- User clicked on the ***List Item*** button from **Main Menu**.
- Run the **List New Item** task:
    - Find out the count of unaccepted swaps from Swap where $Email = proposer_email AND swap_id NOT present in AcknowledgedSwap
    - It the result from this query >2 then
    - Display a popup with an error message.
    - End **List New Item** task.

```
SELECT COUNT(1) unaccepted_swap_count
FROM Swap s
WHERE s.proposer_email = '$Email" and
s.swap_id not in (select swap_id from AcknowledgedSwap)
```

- Find out the count of unrated swaps from Swap where $Email = proposer_email or counterpart_email AND swap_id NOT present in RatedSwap
- It the result from this query >5 then
- Display a popup with an error message.
- End **List New Item** task.

```
((SELECT COUNT(1) unrated_swap_count from Swap s
JOIN AcknowledgedSwap a
ON s.swap_id = a.swap_id
WHERE AcknowledgedSwap.status = 'ACCEPTED'
AND (s.proposer_email = ‘$Email' OR s.counterparty_email = '$Email')
AND a.swap_id NOT IN
(SELECT swap_id from Ratedswap where email = '$Email')));
```

- Go to **Listing an Item** form.
  - User selects a game type from the *Game type* (‘$GameType’) dropdown:

```
SELECT DISTINCT Type AS GameType FROM Item
```

- User enters the 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:

```
INSERT INTO Item(‘email’, ‘name’, ‘description’, ‘condition’, ‘type’,) VALUES ( ‘$Email’,
‘$Name’, ‘$Description’, ‘$Condition’, ‘$Type’)
```

- This insertion results in creating an auto generated item_id.

Table of Contents

```
SELECT LAST_INSERT_ID();
```

- Next we select the GameType and insert into relevant table using the item_id just generated
    - 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.
            - If this is selected as a Type then, the insert will be made to the respective VIDEO_GAME table

```
INSERT INTO VideoGame('item_id', 'platform', 'media') VALUES
('$item_id','$Platform', '$Media')
```

- Else if '$GameType' == "Computer game", then:
    - Show the *Platform* ('$Platform') dropdown.
    - Find platforms for '$GameType' == "Computer Game" and populate the *Platform* dropdown.
    - If this is selected as a Type then, the insert will be made to the respective COMPUTER_GAME table

```
INSERT INTO ComputerGame('item_id', 'platform', ) VALUES('$item_id','$Platform',)
```

- Else if '$GameType' == "Jigsaw puzzle", then:
    - Show the *Piece count* ('$PieceCount') input field.
    - If this is selected as a Type then, the insert will be made to the respective JIGSAW_PUZZLE table

```
INSERT INTO JigsawPuzzle ('item_id', 'piece_count") VALUES ('$item_id', '$PieceCount')
```

- Else if '$GameType' == "BoardGame", then:

                - If this is selected as a Type then, the insert will be made to the respective BoardGame table

```
INSERT INTO BoardGame (‘item_id’,)
VALUES (‘$item_id’)
```

            - Else if ‘$GameType’ == “CardGame", then:
                - If this is selected as a Type then, the insert will be made to the respective CardGame table

```
INSERT INTO CardGame (‘item_id’,)
VALUES (‘$item_id’)
```

            - 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 get added along with an entry in the Type Table depending on the selection of the GameType.
                - Display a success popup with the newly listed ItemID.
    - Else:
        - Display the **<u>Listing an Item</u>** form with error message.

<u>View My Items</u>

Abstract Code

- User clicked on the ***My items*** button from the **<u>Main Menu</u>**.
- 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.

```
SELECT COUNT(1) FROM Item WHERE TYPE = 'BoardGame' AND
email = '$Email'
```

```
SELECT COUNT(1) FROM Item WHERE TYPE = 'CardGame' email =
'$Email'
```

```
SELECT COUNT(1) FROM Item WHERE TYPE = 'ComputerGame'
email = '$Email'
```

```
SELECT COUNT(1) FROM Item WHERE TYPE = 'JigsawPuzzle' AND
email = '$Email'
```

```
SELECT COUNT(1) FROM Item WHERE TYPE = 'VideoGame' WHERE
email = '$Email'
```

```
SELECT COUNT(1) FROM Item WHERE EMAIL = '$Email'  AND TYPE
IN ('BoardGame', 'CardGame', 'VideoGame',
'ComputerGame','JigsawPuzzle')
```

  - 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.

```
SELECT item_id, name, LEFT("description",100),condition, type
FROM Item WHERE email = "$Email" ORDER BY item_id ASC;
```

- Upon click of the ***Detail*** link - Jump to the **View Item** task passing the Item.ItemID as a url parameter showing the details of the item selected

```
SELECT item_id, name, description, condition, type  FROM Item WHERE item_id
= '$item_id;
```

Search for Items

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.

    ```
    SELECT postal_code FROM location
    ```

  - User selects one of the 4 *search criteria* ('$SearchCriteria') radio buttons.
  - User enters either the *By keyword* ('$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 '$Keyword' is null, then:
          - Display the **Searching for Items** form with error message.
        - Else run search with '$Keyword'

          ```
          SELECT item.item_id, item.type, item.name,
          item.condition,item.description,RADIANS( location.latitude)
          Latitude, RADIANS(location.longitude) longitude
          FROM item
          JOIN user ON user.email = item.email
          JOIN location ON user.postal_code = location.postal_code
          WHERE NOT user.email = '$email' AND (item.name LIKE
          '%$Keyword%'
          ```

```
OR item.description LIKE '%$Keyword%')
```

- If the query returns no result display “Sorry, no results found!”
- else
    - Get the ‘$email’ latitude and longitude

      ```
      SELECT RADIANS( location.latitude) Latitude,
      RADIANS(location.longitude) longitude
      FROM location
      JOIN user
      ON user.postal_code = location.postal_code
      WHERE user.email = ''$email'
      ```

    - Get the latitude and longitude for each items found and calculate the distance with the ‘$email’ latitude and longitude
    - Replace latitude and longitude with distance and sort the search items by distance.
    - Distance from the user is rounded to tenth.
    - Highlight item.name or item.description that matched the ‘$Keyword’
    - Add detail link for each items

- Else if ‘$SearchCriteria’ is “Within X miles of me”, then:
    - If ’$DistanceWithin’ is NULL, then:
        - Display the **<u>Searching for Items</u>** form with error message.
    - Else run search with ’$DistanceWithin’

      ```
      SELECT item.item_id, item.type, item.name,
      item.condition, item.description, RADIANS(
      location.latitude) Latitude,
      RADIANS(location.longitude) longitude
      FROM item
      JOIN user
      ON user.email = item.email
      JOIN location
      ON user.postal_code = location.postal_code
      WHERE NOT user.email = '$email'
      ```

- If the query returns no result display “Sorry, no results found!”
- else
    - Get the ‘$email’ latitude and longitude

      ```
      SELECT RADIANS( location.latitude) Latitude,
      RADIANS(location.longitude) longitude
      FROM location
      JOIN user
      ON user.postal_code = location.postal_code
      WHERE user.email = "$email’
      ```

    - Get the latitude and longitude for each items found and calculate the distance in Radians with the ‘$email’ latitude and longitude
    - Replace latitude and longitude with distance and sort the search items by distance.
    - Distance from the user is rounded to tenth.
    - Remove items that are more than ’$DistanceWithin’
    - Add detail link for each items

- Else if ‘$SearchCriteria’ is “In my postal code”, then:
    - Run search with ’$User.PostalCode

      ```
      SELECT item.item_id, item.type, item.name,
      item.condition, item.description,RADIANS(
      location.latitude) Latitude,
      RADIANS(location.longitude) longitude
      FROM item
      JOIN user
      ON user.email = item.email
      JOIN location
      ON user.postal_code = location.postal_code
      WHERE NOT user.email = '$email' AND
      user.postal_code = '$PostalCode'
      ```

    - If the query returns no result display “Sorry, no results found!”
    - else
        - Get the ‘$email’ latitude and longitude

```
SELECT RADIANS( location.latitude) Latitude,
RADIANS(location.longitude) longitude
FROM location
JOIN user
ON user.postal_code = location.postal_code
WHERE user.email = '$email'
```

        ○ Get the latitude and longitude in Radians for each items found and calculate the distance with the '$email' latitude and longitude in Radian.
        ○ Replace latitude and longitude with distance and sort the search items by distance.
        ○ Distance from the user is rounded to tenth.
        ○ Add detail link for each items

- Else if '$SearchCriteria' is "In postal code", then:
    - If '$PostalCode' is NULL, then:
        - Display the **Searching for Items** form with an error message.
    - Else run search with '$PostalCode

```
SELECT item.item_id, item.type, item.name,
item.condition, item.description, RADIANS(
location.latitude) Latitude,
RADIANS(location.longitude) longitude
FROM item
JOIN user
ON user.email = item.email
JOIN location
ON user.postal_code = location.postal_code
WHERE NOT user.email = '$email' AND
user.postal_code = '$PostalCode'
```

    - If the query returns no result display "Sorry, no results found!"
    - else
        - Get the '$email' latitude and longitude

```
SELECT RADIANS( location.latitude) Latitude,
RADIANS(location.longitude) longitude
FROM location
JOIN user
```

```
ON user.postal_code = location.postal_code
WHERE user.email = '$email'
```

        - Get the latitude and longitude in Radian for each items found and calculate the distance with the '$email' latitude and longitude
        - Distance from the user is rounded to tenth.
        - Replace latitude and longitude with distance and sort the search items by distance
        - Add detail link for each items
      - Display only first 100 character from description and place ... to represent continuation.
    - Upon click of the ***Detail*** link - Jump to the **View Item** task passing the Item.ItemID as a url parameter.

View Item

Abstract Code

- Run the **View Item** task:
  - Find Item.ItemID, Item.Name, Item.Type, Item.Condition

```
SELECT item.item_id, item.type, item.name,
item.condition,item.type,item.email
FROM item
WHERE item.item_id = '$Item.ItemID'
```

    - Set item id, item type, item condition and item name with result return from query
  - If return Item.type is Computer Game

```
SELECT platform
FROM computergame
WHERE item_id = '$Item.ItemID'
```

    - Set platform with result return from query
  - Else if return Item.type is jigsaw puzzle

```
SELECT piece_count
FROM jigsawpuzzle
WHERE item_id = '$Item.ItemID'
```

    - Display piece count with result return from query
  - Else if return Item.type is video game

```
SELECT platform,media
FROM videogame
WHERE item_id = '$Item.ItemID'
```

- ■ Display media and platform with result return from query

○ If Item.email is not '$User' then display counterparty information. Store the counterparty postal code as '$CounterpartyPostalCode'

```
SELECT user.nickname, AVG(ratedswap.rating) Rating, RADIANS(
location.latitude) Latitude, RADIANS(location.longitude) longitude,
location.city, location.state, location.postal_code
FROM user
JOIN location ON user.postal_code = location.postal_code
LEFT OUTER JOIN
Ratedswap  ON ratedswap.email = user.email WHERE user.email = '$user'
```

- ■ Get the '$User' latitude and longitude

```
SELECT RADIANS( location.latitude) Latitude,
RADIANS(location.longitude) longitude
FROM location
JOIN user
ON user.postal_code = location.postal_code
WHERE user.email = '$Useremail'
```

- ■ Get the latitude and longitude for other user and calculate the distance with the '$Useremail' latitude and longitude
- ■ Replace latitude and longitude with distance
- ■ If '$CounterpartyPostalCode' and '$User.postal_code' are equal then hide distance
- ■ else
  - Round distance to tenths
  - If the distance between 0.0 and 25.0 miles highlighted with a green background
  - Else if the distance between 25.0 and 50.0 miles highlighted with a yellow background
  - Else if the distance between 50.0 and 100.0 miles highlighted with a orange background
  - Else highlighted with a red background

- Get user unrated swap count

```
((SELECT COUNT(1) unrated_swap_count from Swap s
JOIN AcknowledgedSwap a
ON s.swap_id = a.swap_id
WHERE a.status = 'ACCEPTED'
AND (s.proposer_email = '$User' OR s.counterparty_email = '$User')
AND a.swap_id NOT IN
(SELECT swap_id from Ratedswap where email = '$User')));
```

- Get user unaccepted swap

```
SELECT COUNT(*) FROM swap
JOIN acknowledgedswap ON swap.swap_id =acknowledgedswap.swap_id
WHERE NOT acknowledgedswap.status = 'completed' AND
swap.proposer_email = '$Useremail'
```

- Check if the item is available for swap

```
SELECT COUNT(*) FROM swap
WHERE swap.proposed_item_id = '$Item.item_id' OR
swap.counterparty_item_id = '$Item.item_id'
```

- If the user has less than 2 unrated swaps, or less than five unaccepted swaps, and the item is available for swapping, a "Propose swap" option should be displayed.

Add Proposed Swap

Abstract Code

- User clicked on the ***Propose swap*** button from **View Item**.
- Run the **Add Proposed Swap** task:
    - Get user unrated swap count

```
((SELECT COUNT(1) unrated_swap_count from Swap s
JOIN AcknowledgedSwap a
ON s.swap_id = a.swap_id
WHERE a.status = 'ACCEPTED'
AND (s.proposer_email = '$User' OR s.counterparty_email = '$User')
AND a.swap_id NOT IN
(SELECT swap_id from Ratedswap where email = '$User')));
```

- If unrated swap count is more than 2

        - display an error message saying user has more than 2 unrated swaps without going into the form
    - Else
        - Get the item.name and the location of counterparty that owns Item

```
SELECT item.name, item.email, item.item_id, RADIANS(
location.latitude) Latitude, RADIANS(location.longitude) longitude
 FROM item
JOIN user ON user.email = item.email
JOIN location ON user.postal_code = location.postal_code
WHERE item.item_id = '$item.item_id'
```

        - Store item.email as '$CounterpartyEmail' and item.item_id as '$CounterpartyItemId'
        - Get the '$User' latitude and longitude

```
SELECT RADIANS( location.latitude) Latitude,
RADIANS(location.longitude) longitude FROM location
JOIN user ON user.postal_code = location.postal_code
WHERE user.email = '$UserEmail'
```

        - Get the latitude and longitude for counterparty and calculate the distance with the '$UserEmail' latitude and longitude
        - If the counterparty is >= 100.0 miles from the user, a warning message containing that distance should be shown at the top of the form
        - List all the items that from '$UserEmail' that is not part of the swap for user to propose the swap

```
SELECT item.item_id, item.type, item.name, item.condition FROM
Item
LEFT OUTER JOIN
swap ON (counterparty_item_id = item_id)
LEFT OUTER JOIN Acknowledgedswap ON (swap.swap_id =
Acknowledgedswap.swap_id)
WHERE (counterparty_email = '$user' AND
Acknowledgedswap.status = 'REJECTED')
OR (item.email = '$user' AND swap.swap_id IS NULL)
```

- Once user select the item then store item.item_id as '$UserProposedItemId' add an entry in the swap table

```
INSERT into Swap (proposer_email, counterparty_email,
proposed_item_id, counterparty_item_id, proposed_date)
 VALUES ('$Useremail', '$CounterpartyEmail',
'$UserProposedItemId', '$CounterpartyItemId', CAST(now() As
Date))
```

Accept/Reject Swap

Abstract Code

- User clicked on the ***Unaccepted swaps*** link from **Main Menu**.
- Run the **Accept/Reject Swap** task:
    - Get all the pending accept or reject swaps

```
SELECT Swap.proposed_date, User.nickname proposer, item1.name
proposed_item ,item2.name desired_item, RADIANS( location.latitude) Latitude,
RADIANS(location.longitude) longitude,
(SELECT AVG(rating) Rating FROM ratedswap WHERE email=proposer_email
GROUP BY email) Rating
FROM User
JOIN Swap ON (Swap.proposer_email = User.email)
LEFT OUTER JOIN Item AS item1 ON (swap.proposed_item_id = item1.item_id)
LEFT OUTER JOIN Item AS item2 ON (swap.counterparty_item_id =
item2.item_id)
LEFT OUTER JOIN Location ON (User.postal_code = Location.postal_code)
WHERE swap.swap_id IN
(SELECT swap_id from swap where counterparty_email = '$user' AND swap_id
NOT IN (SELECT Acknowledgedswap.swap_id from swap JOIN AcknowledgedSwap
ON swap.swap_id = Acknowledgedswap.swap_id WHERE counterparty_email =
'$user'));
```

- Get the '$User' latitude and longitude

```
SELECT RADIANS( location.latitude) Latitude, RADIANS(location.longitude)
longitude
FROM location
JOIN user ON user.postal_code = location.postal_code
WHERE user.email = '$Useremail'
```

- ○ Get the latitude and longitude for the query and calculate the distance with the '$User' latitude and longitude
- If the user accepts the proposed swap
    - Run the **Accept** task:
        - With the Swap.proposed_item_id from the selected swap get the Swap.swap_id and store as '$SwapId'

```
SELECT swap_id FROM swap
WHERE proposed_item_id = '$Swap.proposed_item_id'
```

        - Update Acknowledgedswap with '$Swap_id'

```
INSERT INTO acknowledgedswap
VALUES ( '$SwapId', 'ACCEPTED', CAST(now() As Date))
```

        - Prompt accept message
            - With $Swap_id get the swap.proposer email, user.phone_number and user.share_phone_number

```
SELECT user.email, user.phone_number, user.share_phone_number
FROM user
JOIN swap ON user.email = swap.proposer_email
WHERE swap.swap_id = '$SwapId'
```

            - If user.share_phone_number is false, prompt accept message without sharing the phone number
            - Else prompt accept message and share the phone number
- Else the user reject the swap
    - Run the reject task:
        - With the Swap.proposed_item_id from the selected swap get the Swap.swap_id and store as '$SwapId'

```
SELECT swap_id FROM swap
WHERE proposed_item_id = '$Swap.proposed_item_id'
```

        - Update AcknowledgeSwap with '$SwapId'

```
INSERT INTO acknowledgedswap
VALUES ( '$SwapId', 'REJECTED', CAST(now() As Date))
```

- Run the **Accept/Reject Swap** task again
- If there are no more item
    - Go back **Main Menu** page.

View Unrated Swaps

Abstract Code

- User clicked on the ***Unrated swaps*** link from **Main Menu**.
- Run the **View Unrated Swaps** task:
    - Find Swaps where the Swap.Proposer_email is '$User' or Swap.Counterparty_email is '$User' and the status is 'ACCEPTED' but Ratedswap does not contains rating for the '$User' and '$swap_id'

```
SELECT A.swap_id, A.acknowledged_date AS Acceptance_Date,
(SELECT COUNT(*) FROM Swap where swap_id = A.swap_id AND proposer_email =
'$user') My_Role,
(SELECT name FROM Item where Item_id = S.proposed_item_id) AS Proposed_Item,
(SELECT name FROM Item where Item_id = S.counterparty_item_id) AS Desired_Item,
(SELECT nickname FROM User where (email != '$user'
AND (email = S.counterparty_email OR email = S.proposer_email))) AS Other_User,
R.rating
 FROM Swap AS S
 JOIN Acknowledgedswap AS A ON S.swap_id = A.swap_id
 LEFT OUTER JOIN Ratedswap AS R ON (R.swap_id = A.swap_id AND R.email = '$user')
 WHERE S.Swap_id
IN
((SELECT Swap.swap_id from Swap
JOIN Acknowledgedswap
ON Swap.swap_id = Acknowledgedswap.swap_id
WHERE Acknowledgedswap.status = 'ACCEPTED'
AND (Swap.counterparty_email = '$user' OR  Swap.proposer_email = '$user')
AND Acknowledgedswap.swap_id NOT IN
(SELECT swap_id from Ratedswap where email = '$user')))
ORDER BY acknowledged_date DESC;
```

- Display a table with the Swap.Acknowledge_date, 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:
        - Jump to the **Swap Rating Update** task with the following

- '$SwapID' = Swap.SwapID
- '$email' = User email
- '$Rating'

Swap Rating Update

Abstract Code

- User selected a rating from the *Rating* dropdown from **Swap History, Rate Swaps or Swap Details**.
    - Update the Rating of the Ratedswap record with Swap.SwapID = '$SwapID' and email = '$email' to the value '$Rating'.

```
INSERT into RatedSwap (swap_id,email,rating)
VALUES ('$swap_id','$email', '$rating');
```

View Swap History

Abstract Code

- User clicked on ***Swap History*** button from **Main Menu**:
- Run **Swap History** task:
- Find Swaps where either Swap.Proposer_email is '$User' or Swap.Counterparty_email is '$User', sorted by Swap.Acknowledged_date descending and Swap.Proposed_date ascending.

```
SELECT S.proposed_date, A.acknowledged_date AS Accept_Reject_Date, A.status,
(SELECT name FROM Item WHERE Item_id = S.proposed_item_id) AS Proposed_Item,
(SELECT name FROM Item WHERE Item_id = S.counterparty_item_id) AS Desired_Item,
(SELECT COUNT(*) FROM Swap where swap_id = A.swap_id AND proposer_email =
'$user') My_Role,
(SELECT nickname FROM User WHERE (email != '$user'
AND (email = S.counterparty_email OR email = S.proposer_email))) AS Other_User,
R.rating FROM Swap AS S
JOIN AcknowledgedSwap AS A ON S.swap_id = A.swap_id
LEFT OUTER JOIN RatedSwap AS R ON A.swap_id = R.swap_id AND R.email='$user'
WHERE (S.counterparty_email = '$user' OR  S.proposer_email = '$user')
ORDER BY A.acknowledged_date DESC, S.proposed_date ASC;
```

- If Swap.proposer_email is '$User' and for all '$swap_id' then:

```
SELECT COUNT(*) FROM Swap where swap_id = '$swap_id' AND proposer_email =
'$user';
```

  If count is 1 : Then My role is proposer
  If count is 0: Then My role is counterparty
- From the results compute the following statistics:
  - Count the number of Swap proposed by user '$User' by checking Swap.ProposerEmail = '$User'

```
SELECT COUNT(*) from AcknowledgedSwap
WHERE swap_id
IN (SELECT swap_id from Swap WHERE Swap.proposer_email = '$user');
```

- Count the number of accepted Swaps that were proposed by user '$User' by checking Swap.ProposerEmail == '$User' and Swap.Status == 'ACCEPTED'

```
SELECT COUNT(*) from AcknowledgedSwap WHERE swap_id
IN (SELECT swap_id from SWAP WHERE Swap.proposer_email = '$user')
AND status = 'ACCEPTED';
```

- Count the number of rejected Swaps that were proposed by user '$User' by checking Swap.ProposerEmail = '$User' and Swap.Status = 'REJECTED'

```
SELECT COUNT(*) from AcknowledgedSwap WHERE swap_id
IN (SELECT swap_id from SWAP WHERE Swap.proposer_email = '$user')
AND 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% and if Rejected_percentage is >=50% background is in Red color.

```
SELECT count(*) * 100.0 / (SELECT COUNT(*)  Rejected% from Swap WHERE
Swap.proposer_email = '$user') Rejected_percentage
FROM AcknowledgedSwap
WHERE swap_id
IN (SELECT swap_id from Swap WHERE Swap.proposer_email = '$user')
AND status = 'REJECTED';
```

- Count the number of Swaps proposed to user '$User' by checking Swap.CounterpartyEmail == '$User'

```
SELECT COUNT(*) from AcknowledgedSwap WHERE swap_id
IN (SELECT swap_id from Swap WHERE Swap.counterparty_email = '$user');
```

- Count the number of accepted Swaps that were proposed to user '$User' by checking Swap.CounterpartyEmail == '$User' and Swap.Status == 'ACCEPTED'

```
SELECT COUNT(*) from AcknowledgedSwap WHERE swap_id
IN (SELECT swap_id from Swap WHERE Swap.counterparty_email = '$user')
AND status = 'ACCEPTED';
```

- Count the number of rejected Swaps that were proposed to user '$User' by checking Swap.CounterpartyEmail == '$User' and Swap.Status == 'REJECTED'

```
SELECT COUNT(*) from AcknowledgedSwap WHERE swap_id
IN (SELECT swap_id from Swap WHERE Swap.counterparty_email = '$user')
AND 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%. if Rejected_percentage is >=50% background is in Red color.

Table of Contents

```
SELECT count(*) * 100.0 / (SELECT COUNT(*) from Swap WHERE
Swap.counterparty_email = '$user') Rejected_percentage
FROM AcknowledgedSwap WHERE swap_id
IN (SELECT swap_id from Swap
WHERE Swap.counterparty_email = '$user') AND status = 'REJECTED';
```

- Display the swap statistics for user '$User' in a table in the UI.
- For each Swap in the UI as a row in a table:
    - Display *Proposed Date* with Swap.proposed_date
    - Display *Accepted/Rejected Date* with AcknowledgedSwap.acknowledged_date
    - Display Swap status with AcknowledgedSwap.status
    - If Swap.proposer_email is '$User', then:
        - Display *My role* with 'Proposer'

      Else
        - Display *My role* with 'Counterparty'
    - If RatedSwap.rating is exist for given RatedSwap.swap_id,and RatedSwap.email = '$email' then:
        - Display *Rating* with RatedSwap.rating
    - Else display a ('$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:
                - '$SwapID' = Swap.SwapID
                - '$email' = User email
                - '$Rating'

- Find proposed item in Item using Swap.proposed_itemid; Display *Proposed Item* with Item.name

```
SELECT name FROM Item WHERE Item_id = S.proposed_item_id
```

- Find desired item in Item using Swap.counterparty_itemid; Display *Desired Item* with Item.name

```
SELECT name FROM Item WHERE Item_id = S.counterparty_item_id
```

- Find other user in User using Swap.CounterPartyEmail; Display *Other User* with User.Nickname

```
(SELECT nickname FROM User WHERE (email != '$user'
AND (email = S.counterparty_email OR email = S.proposer_email))) AS Other_User
```

- 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

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 $Swap_id provided in the url parameter.

```
SELECT S.proposed_date, A.acknowledged_date, A.status, R.rating
FROM Swap AS S
JOIN AcknowledgedSwap AS A ON S.swap_id = A.swap_id
LEFT OUTER JOIN RatedSwap AS R ON (R.swap_id = A.swap_id AND R.email ='$user')
WHERE s.swap_id = '$swap_id';
```

- Display the *Proposed* with Swap.proposed_date, *Accepted/Rejected* with Acknowledgedswap.acknowlegded_date, *Status* with Acknowledgedswap.status.
- If Swap.proposer_email is '$User' then:

```
SELECT COUNT(*) FROM Swap where swap_id = '$swap_id' AND proposer_email =
'$user';
```

If count returns 1:
  - Display *My role* as 'Proposer'

If count returns 0:
  - Display *My role* as 'Counterparty"
  - If Ratedswap.rating is not null for the $SwapID and $user, then:
    - Display *Rating left* with Ratedswap.rating
  - Else display a ('$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

        - '$email = user email
        - '$Rating'

    - Find Other User invloved in swap using not current user $email and$swap_id

```
SELECT User.first_name, User.nickname, User.email, P.phone_number, P.type,
P.share_phone_number, RADIANS( location.latitude) Latitude, RADIANS(location.longitude)
longitude FROM User
JOIN Swap ON (Swap.counterparty_email = User.email
OR Swap.proposer_email = User.email)
JOIN PhoneNumber AS P ON P.email = User.email
JOIN Location ON (User.postal_code = Location.postal_code)
WHERE email NOT IN ('$user') and Swap.swap_id = '$swap_id';
```

        - Display *Nickname* with proposer User.nickname, *Name* with User.FirstName, *email* with User.Email and *phone* with User.PhoneNumber
        - Find current user '$User' location in Location

```
SELECT RADIANS( location.latitude) Latitude, RADIANS(location.longitude) longitude
 from User
JOIN Location ON (User.postal_code = Location.postal_code)
AND User.email = '$User';
```

            - Use Location.Latitude and Location.longitude to calculate distance to user '$User' and display as *Distance*.

- Find Item involved in swap using current user $email and $swap_id

```
SELECT Item.item_id,Item.name,Item.type,Item.condition,Item.description from Item
JOIN Swap ON (Swap.proposed_item_id = Item.item_id
OR Swap.counterparty_item_id = Item.item_id)
WHERE Item.email = '$user' AND Swap.swap_id = '$swap_id';
```

    - 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 Other Item involved in swap using not current user $email and $swap_id

```
SELECT Item.item_id,Item.name,Item.type,Item.condition,Item.description from Item
JOIN Swap ON (Swap.proposed_item_id = Item.item_id
OR Swap.counterparty_item_id = Item.item_id)
WHERE Item.email NOT IN ('$user') and Swap.swap_id = '$swap_id';
```

- 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 My Info

Abstract Code

- User clicked on the ***Update my info*** button from **Main Menu**
- Run the **Update User Information** task:
    - Find swap_id which are in Swap where the Swap.ProposerEmail or Swap.CounterPartyEmail is $User and the status is present in AcknowledgedSwap but not in RatedSwap entity
      (Swaps which are ACCEPTED / REJECTED but have not yet been rated)

```
((SELECT COUNT(1) unrated_swap_count from Swap s
JOIN AcknowledgedSwap a
ON s.swap_id = a.swap_id
WHERE AcknowledgedSwap.status = 'ACCEPTED'
AND (s.counterparty_email = '$Email' OR s.proposer_email = '$Email')
AND AcknowledgedSwap.swap_id NOT IN
(SELECT swap_id from Ratedswap where email = 'user4@gatech.edu')))
```

- If the occurrence is more than zero; Display a popup with an error message.
  IF COUNT(1)>0 display appropriate error message

    - Find swap_id which are in Swap where the Swap.ProposerEmail or Swap.CounterPartyEmail is $User and the swap_id is not present in AcknowledgedSwap

```
SELECT COUNT(1) unaccepted_swap_count
FROM Swap s
WHERE s.proposer_email = '$Email" and
s.swap_id not in (select swap_id from AcknowledgedSwap)
```

        - If the occurrence is more than zero; Display a popup with an error message.
          IF COUNT(1)>0 display appropriate 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)

```
SELECT * FROM User WHERE email = '$Email'
```

- 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

```
SELECT * FROM Location WHERE postal_code = (SELECT postal_code
FROM User WHERE email = '$Email')
```

- 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'

```
UPDATE User SET first_name = '$FirstName', last_name = '$LastName', nickname =
'$NickName', password = '$Password', postal_code= '$PostalCode' WHERE email =
'$Email'
```

```
UPDATE PhoneNumber SET phone_number = '$PhoneNumber', type = '$Type',
share_phone_number = '$PhoneNumber' WHERE email = '$Email'
```

- If the PhoneNumber already exists, appropriate error message is thrown.