**Algorithms**

**Data Retrieval**

1. **Initialization:**
   1. Start the process.
   2. Initialize system databases (Product Database, User Database, Admin Database, Store Database, and Flyer Database).
   3. Retrieve the user's location (for nearby store detection) and logged-in status (User/Admin).
   4. Request user input for product search (keywords, categories, store names, or flyer search).

1. **Fetch Product Details (User Request):**
   1. User searches for a product or selects an item from a grocery list.
   2. Query the **Product Database** based on the search keywords or selected item.

* **Check:** If the product exists in the database.
  + **Yes:** Proceed to the next step.
  + **No:** Display an error message that the product is unavailable.
  1. Retrieve real-time product prices from the **Store Database** for all stores near the user’s location.
* Compare prices across stores.
* Display the list of stores with corresponding prices.

1. **Fetch Inventory Details:**
   1. Once the product is identified, query the **Store Database** for inventory availability.
      * Check: If the product is available in the inventory of the selected stores.
      * **Yes:** Display available quantity for each store.
      * **No:** Notify the user of stock unavailability.
   2. If inventory exists, fetch additional details such as **product quantity offered for price matching**.
2. **Fetch Flyer Details:**
   1. Query the **Flyer Database** for the current flyer information related to the product or store.

* Retrieve details such as:
  + Flyer starts and end dates.
  + Flyer-specific prices or discounts.
  1. Display flyer details, including price validity period (start and end dates) and promotional pricing (if applicable).

1. **Admin-Specific Data Fetch (Admin Request):**
   1. If logged in as **Admin**, query additional admin-specific data such as:

* Current flyer status for the selected store.
* Existing product categories for updating or deleting.
* Price matching rules across stores.
  1. Retrieve current products and store information for **adding, updating, or deleting** items.
* Allow admins to modify price matching details, product quantities, and store records.

1. **User-Specific Data Retrieval:**
   1. the **User Database** for logged-in users to fetch:

* User account details.
* Personalized grocery list.
* Past searches or purchase history.
  1. Fetch **notifications** related to product status (availability, price drop, flyer start date, etc.).
  2. Display personalized recommendations based on current flyer data, inventory, and user preferences.

1. **End Process:**
   1. Finalize and display all retrieved information (prices, inventory, flyers, personalized notifications).
   2. End.

**Cart Management and Checkout**

1. **Add Item to Cart**
2. Begin
3. Search for the item by name
4. If the item exists in the inventory, then
   * If the item already exists in the cart, then
     + Update the quantity or cancel
   * If the item does not exist in the cart, then
     + Add the item to the cart with the specified quantity
5. Recalculate the total amount in the cart every time an item is added
6. End
7. **Update Quantity in Cart**
8. Begin
9. Select the item, then
   * Update the item quantity or cancel
10. Recalculate the total amount in the cart once the item is updated
11. End
12. **Delete Item from Cart**
13. Begin
14. Click on the “trash” button
15. Remove the item from the cart.
16. Recalculate the total amount in the cart once the item is deleted
17. End
18. **Checkout Process**
19. Begin
20. Ask the user to confirm the order
21. If the user confirms, then
    * Proceed to the order gateway
    * Complete the transaction
22. Send a confirmation to the user
23. Clear the cart.
24. End

**Login Authentication Process**

**User Login Authentication**

1. Start
2. For user input ask user to enter email and password.
3. Validation of email format:
   1. If the email format is invalid (e.g., missing "@" or domain):
      1. Display an error message: "Invalid email format."
      2. Return to step 2 to prompt for email and password again.
4. Writing a query in database
   1. Now we will execute the query from the database so that we can retrieve the user details using the provided email.
      1. User= database.query(“select \* from users where email = email”)
5. Checking if User exists or not
   1. If user is NULL (user not found):
      1. Display an error message: "User not found."
      2. Return to step 2 to prompt for email and password again.
6. Verify Password:
   1. Compare the provided password with the stored hashed password:
      1. If user.password is equal to the hash of the provided password:
         1. Set the user session as authenticated (e.g., session.setAuthenticated(user)).
         2. Redirect to the user dashboard (e.g., redirect to dashboard).
      2. Else
         1. Display an error message: "Invalid password."
         2. Return to step 2 to prompt for email and password again.
7. End

**Budget Management**

**Budget Check Feature**

1. Start the Budget Check Process

* Triggered when the user adds or updates an item in the grocery list.

1. Calculate the Total Cost of the List

* Initialize a variable total\_cost = 0.
* For each item in the grocery list:

Multiply the quantity by the price per unit to get the item’s total cost.  
Add the item’s cost to total\_cost.

1. Compare Total Cost with Budget

* If total\_cost > budget\_limit:

Proceed to Step 4 (Notify User).

* Else:  
  Skip to Step 7 (Save List).

1. Notify the User

* Display a warning message: "Budget exceeded. Please adjust your list."

1. Provide Editing Options

* Allow the user to:

Edit item quantity

Remove items

* If the user makes changes, proceed to Step 6.

1. Recalculate the Total Cost

* Recompute the total cost using the updated list.
* Return to Step 3 (Check if the new total is within the budget).

1. Save the List

* If the total cost is within the budget, save the grocery list.

1. End the Budget Check Process

* Display a success message: "List saved successfully."