1. Introduction

This document provides a detailed explanation of the Grocery App project implemented in Python. The application simulates a shopping experience with features like cart operations, saving items for later, generating invoices, storing transactions and feedback using JSON files.

2. Why JSON is Used Instead of CSV

- JSON supports nested and structured data (e.g., dictionary of items with product IDs and quantities).
- It allows easy serialization and deserialization of complex Python objects.
- Better suited for hierarchical data like transactions and feedback with timestamps.
- Easier integration with web-based or API-driven systems (JSON is native to JavaScript).
- CSV is flat and suitable only for tabular data; it lacks support for nested structures.

3. JSONHandler Class

- initialize_json_files(): Creates 'products.json', 'pricing.json', 'transactions.json', and 'feedback.json' if they don't exist.
- It uses `@staticmethod` because the methods don't require access to instance-level data.
- Initializes empty lists for `transactions.json` and `feedback.json`.
- read_products(): Reads product ID and name from products.json.
- **Purpose:**
  Reads the `products.json` file and returns it as a Python dictionary.

How it works:

- Loads JSON data into a dictionary.
- Converts keys from strings to integers using a dict comprehension.

- read_pricing(): Reads product ID and price from pricing.json.
- **Purpose:**
  Reads `pricing.json` and returns a dictionary with product prices.

How it works:

- Loads pricing data from JSON.
- Ensures keys are integers and values are converted to floats.

-

- save_transaction(): Appends transaction details including items, total, and timestamp to transactions.json.

- **Purpose:**
  Saves a completed shopping transaction into `transactions.json`.
What it stores:

- `transaction_id`: Unique UUID string.
- `items`: A dictionary of product ID to quantity (e.g., `{1: 2, 4: 1}`)
- `total`: Total amount of the transaction.
- `timestamp`: Current date and time.

Why it's important:
Maintains a **record of all purchases**, which can be used for analysis or audit.

-

- save_feedback(): Appends user feedback with transaction ID and timestamp to feedback.json.

- **Purpose:**
  Stores customer feedback related to a transaction in `feedback.json`.
What it stores:

- `transaction_id`: Links feedback to a specific transaction.
- `feedback`: User's input (string).
- `timestamp`: Time when feedback was submitted.

Why it's important:
Allows the system to **track user satisfaction or complaints**, tied to specific purchases.

-

- display_json_contents(): Displays all four JSON file contents in a readable format.
Purpose:
Prints the contents of all four JSON files (`products`, `pricing`, `transactions`, `feedback`) in a **readable tabular format** on the console.
Sections displayed:

- **Products**:
  - Format: `Product ID | Name`
- **Pricing**:

    - Format: `Product ID|Price`

- **Transactions**:
    - Format: `Transaction ID|Items|Total|Timestamp`
    - Items are printed as `1:2;3:1` (product_id:quantity)
- **Feedback**:
    - Format: `Transaction ID|Feedback|Timestamp`

Why it's important:
Provides a **quick visual overview** of the current state of the system for debugging or reporting.

4. Cart Class

1. `__init__(self)`

Purpose:
Initializes an empty cart with:

- `items`: Products added to the cart
- `saved_for_later`: Products the user saved to buy later

- add_item(): Adds a specified quantity of a product to the cart.
- **Purpose:**
Adds a product and quantity to the cart.

Logic:

- Checks if `product_id` exists in the available products.
- Adds the quantity to the existing quantity in the cart (or initializes it).

-

- update_item(): Updates the quantity or removes item if quantity is zero.

- **Purpose:**
Updates the quantity of an item in the cart.

Logic:

- Validates the product ID.
- If quantity is 0 or negative, removes the item.

- Otherwise, sets the new quantity.

-

- remove_item(): Deletes an item from the cart.

- **Purpose:**
  Removes a product from the cart completely.
Logic:

- Deletes the key from `self.items` if it exists.

-

- save_for_later(): Moves items from cart to saved list.

- **Purpose:**
  Moves some or all quantity of a product from the cart to a "saved for later" list.
Logic:

- Checks if the product exists in cart and in available products.
- Ensures enough quantity is in the cart.
- Moves quantity from `self.items` to `self.saved_for_later`.

- move_to_cart(): Moves items back from saved list to cart.

- **Purpose:**
  Moves some or all quantity of a saved item back into the cart.
Logic:

- Checks if the product exists in `saved_for_later` and product list.
- Transfers specified quantity back to `items`.
- Reduces or removes the saved quantity accordingly.

-

- view_saved_items(): Displays all items saved for later.

- **Purpose:**
  Displays the list of items the user has saved for later.
Logic:

- Iterates over `saved_for_later`.
- Prints name and quantity of each item.

-

- get_total(): Calculates total cost using pricing.

- **Purpose:**
  Calculates the **total bill amount** based on quantities in the cart and prices.

Logic:

- Iterates through `self.items`.
- Multiplies each product's quantity by its price from the pricing dictionary.
- Sums up the total.

5. InvoiceGenerator Class

This class focuses solely on displaying a bill receipt (invoice) at the end of a transaction.
- generate_invoice(): Prints formatted receipt with item name, quantity, price, total and timestamp.
- **Purpose:**
  Prints a formatted invoice showing:
  - Transaction ID
  - Product names, quantities, price per unit, and total per item
  - Grand total
  - Timestamp

What it does:

1. Prints a heading and transaction ID.
2. Iterates over each item in the cart:
   - Retrieves product name using `product_id` from `products`.
   - Gets the price per unit from `pricing`.
   - Calculates `item_total = price * quantity`.
3. Accumulates the total amount.
4. Displays each row with:

```
ITEM                QUANTITY    PRICE       TOTAL
potato              2           ₹2.00       ₹4.00
```

5. Prints the grand total and current timestamp.

-

6. PaymentProcessor Class

This class is responsible for calculating the final amount to be paid, based on the contents of the cart.
- process_payment(): Calculates the final bill amount by calling cart.get_total().

Purpose:
Returns the total cost of all items in the cart by delegating to `cart.get_total()`.
How it works:

- Calls `cart.get_total(pricing)`
- Returns the total amount as a float

Why use a separate class?

- Follows **Single Responsibility Principle** – in case future logic involves discounts, taxes, or coupons, this class can handle those enhancements.

7. FeedbackManager Class

This class is responsible for asking users for feedback after a transaction and storing it.

- collect_feedback(): Prompts user to give feedback after transaction, and stores it in feedback.json.

Purpose:

Prompts the user to provide feedback related to their transaction and stores it in `feedback.json`.

Steps:

1. Asks: "Do you want to provide feedback? (yes/no)"
2. If user enters "yes":
    - Prompts: "Please enter your feedback"
    - Captures and stores the feedback with timestamp and transaction ID using:

```
python
Copy code
JSONHandler.save_feedback(transaction_id, feedback)
```

    - Shows confirmation: `"Feedback saved successfully!"`
3. If user enters "no":
    - Displays `"No feedback provided."`

Why it's important:

- Helps collect customer opinions.
- Allows later analysis for customer satisfaction.

8. GroceryApp Class

This class is the main controller of the entire Grocery App, handling user interactions and coordinating different components.

- __init__(): Initializes the app with cart, UUID, and loads product & pricing data.
- **Purpose:** Initializes the application by:
    1. Creating an empty shopping cart (`Cart()`).
    2. Generating a unique transaction ID using `uuid.uuid4()`.
    3. Initializing all required JSON files if they don't already exist (`products.json`, `pricing.json`, etc.).

4. Reading products and prices from JSON files.

- start(): Main loop for taking user actions (add, remove, update, view, save, move, done).

- **Purpose:** Starts the user interaction loop.
Steps:

1. Greets the user and displays all available products with their prices.
2. Repeatedly prompts the user for actions:
    - `add`, `update`, `remove`, `save`, `move`, `view`, `done`
3. Based on the input:
    - Calls appropriate helper methods (`_handle_add`, etc.)
4. If the user enters `done`:
    - Saves the transaction
    - Displays an invoice
    - Collects feedback
    - Shows contents of all JSON files
    - Exits the app

- _handle_add(): Handles adding items to cart.

- **Purpose:** Handles the `"add"` action.
Steps:

1. Asks for input like: `1,3` (product ID 1, quantity 3).
2. Parses it into `product_id` and `quantity`.
3. Calls `self.cart.add_item(...)`.
4. Displays a success or error message.

- _handle_update(): Handles updating quantity in cart.

- **Purpose:** Updates the quantity of an item in the cart.
Steps:

1. Asks for input: `product_id,quantity`
2. Parses and validates

3. Calls `cart.update_item(...)`
4. If quantity is `0`, item is removed.

- _handle_remove(): Removes item from cart.

- **Purpose:** Removes an item from the cart completely.
Steps:

1. Asks for a product ID.
2. Converts it to `int` and removes it using `cart.remove_item(product_id)`
3. Displays confirmation.

- _handle_save_for_later(): Saves cart item for later.

- **Purpose:** Moves a product from the cart to the "Saved for Later" list.
Steps:

1. Asks for input like: `1,2` (save 2 potatoes for later)
2. Parses and validates it
3. Calls `cart.save_for_later(...)`
4. Displays confirmation or error

- _handle_move_to_cart(): Moves item from saved-for-later to cart.

- **Purpose:** Moves a product back from "Saved for Later" into the cart.
Steps:

1. Asks for input: `product_id,quantity`
2. Parses and validates
3. Calls `cart.move_to_cart(...)`
4. Shows confirmation or error

```
if __name__ == "__main__":
    app = GroceryApp()
    app.start()
```

Purpose:

This is the entry point. It:

Creates an instance of GroceryApp

Calls start() to begin the application loop