Programming Project

(Minh Tri, Alphonso, Pyae Shan)

Click here to View Project Repository

Given Objective:

To develop a Python-based fast-food ordering and payment application that allows users to:

- View the menu
- Choose items across various categories
- Customize their selections, and
- Complete the payment process.

The system incorporates discount options and automatically computes the total bill, including GST.

User's Needs

- Simple menu browsing by category
- Easy item selection and quantity change
- Cart management
- Combo meal editing
- Discount application (student, staff, loyalty member)
- GST and discount calculation with receipt output

So, we are currently helping a restaurant make its ordering system more efficient for users.

Restaurant name: Obama Fried Chicken

Restaurant motto:

‘Affordable and customizable dining service for all customers’

P.S. This restaurant has been shut down in China due to a complaint for mocking the President. Now, it has been renamed as UFO.

Menu Categories

🍔 Burgers	🍟 Sides	🥤 Drinks	🍦 Desserts	🍱 Combos

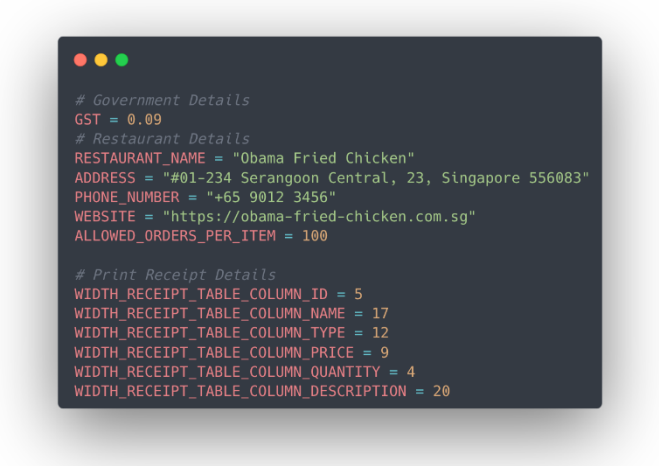
Discount Eligibility	
Student	10%
Staff	8%
Loyalty Member	5%

Data Structure Design

This is the current dataset for our food menu.

The structure utilizes a well-structured data organization approach:

Constant variables (can be changeable):



```
# Government Details
GST = 0.09
# Restaurant Details
RESTAURANT_NAME = "Obama Fried Chicken"
ADDRESS = "#01-234 Serangoon Central, 23, Singapore 556083"
PHONE_NUMBER = "+65 9012 3456"
WEBSITE = "https://obama-fried-chicken.com.sg"
ALLOWED_ORDERS_PER_ITEM = 100

# Print Receipt Details
WIDTH_RECEIPT_TABLE_COLUMN_ID = 5
WIDTH_RECEIPT_TABLE_COLUMN_NAME = 17
WIDTH_RECEIPT_TABLE_COLUMN_TYPE = 12
WIDTH_RECEIPT_TABLE_COLUMN_PRICE = 9
WIDTH_RECEIPT_TABLE_COLUMN_QUANTITY = 4
WIDTH_RECEIPT_TABLE_COLUMN_DESCRIPTION = 20
```

Dataset:

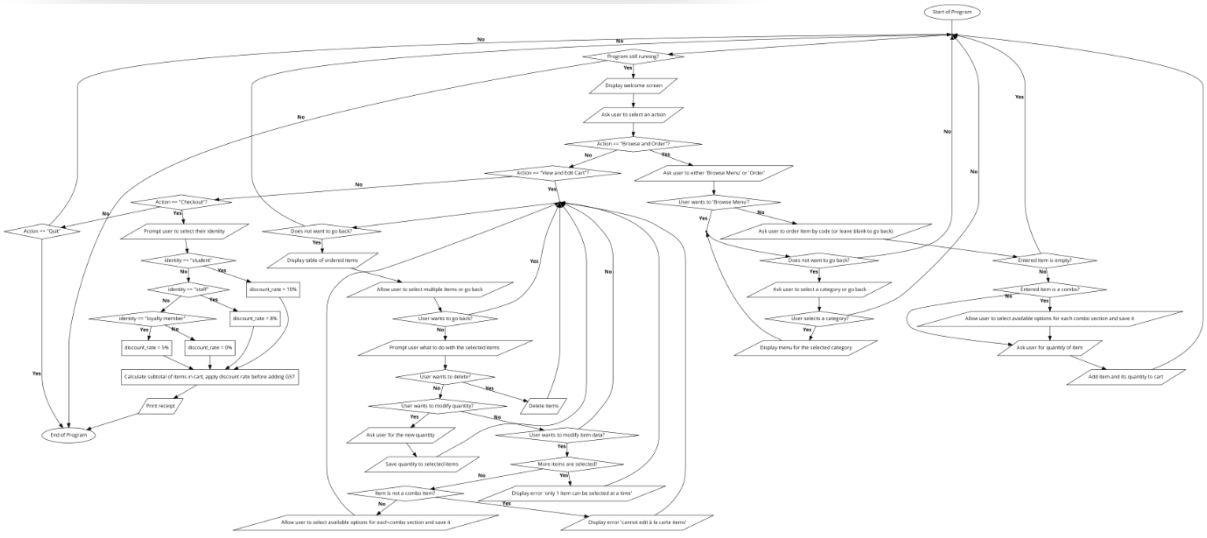
```python
# { id: (noun_name, plural_name, icon) }
MENU_ITEM_IDS = {
    "B": ("Burger", "Burgers", "🍔"),
    "S": ("Side", "Sides", "🍟"),
    "D": ("Drink", "Drinks", "🥤"),
    "DS": ("Dessert", "Desserts", "🍦"),
    "C": ("Combo", "Combos", "🥡")
}

MENU = [
    # Burgers
    {"id": "B01", "name": "Classic Beef Burger", "price": 5.50},
    ...
    # Sides
    {"id": "S01", "name": "French Fries", "price": 2.50},
    ...
    # Drinks
    {"id": "D01", "name": "Coke", "price": 1.80},
    ...
    # Desserts
    {"id": "DS01", "name": "Chocolate sundae", "price": 2.80},
    ...
    # Combos (for item reference ids, just only the alphabet id = any)
    {
        "id": "C01",
        "name": "Burger + Fries + Drink",
        "item_ref_ids": {
            "Burger": (["B"], 1),
            "Fries": (["S01"], 1),
            "Drink": (["D"], 1)
        },
        "price": 8.80
    },
    ...
]

DISCOUNT_RATES = {
    "student": 0.10,
    "staff": 0.08,
    "loyalty_member": 0.05
}
```

A Walkthrough of our program

Click here to View Flowchart



Our flowchart shows a brief and detailed flow of how our ordering system works.

In our code, we have separated each part of our program into different components.

Flowchart Analysis

Based on the flowchart, the system follows a structured workflow:

1. **Initialization:** Displays welcome screen and main menu options
2. **Menu Browsing:** Allows user to browse categories and view items
3. **Order Building:** Enable item selection with quantity specification
4. **Cart Management:** Provide options to edit, remove, or modify cart contents
5. **Checkout Process:** Calculate total, apply discounts, and generate receipt
6. **Order Completion:** Display final receipt and order confirmation

Core Functionality

Our systems are carefully designed to make food ordering as user-friendly as possible in command-line interfaces.

1. Menu Management

The system provides menu item handling with functions for:

- Retrieving items by each category
- Finding items by their unique ID
- Supporting combo meals with item reference IDs, or just the category code to reference all items in that category
- Generating formatted item tables for display

2. Order processing

Key features include:

- Parsing, validation, and combo meal sections with auto-lock features for combo meals
- Real time subtotal calculation for cart
- Convenient “Back” feature to navigate back and edit data

3. Receipt Generation

The receipt system offers:

- GST calculation
- Applying discounts based on the user's selected identity
- Comprehensive order summary

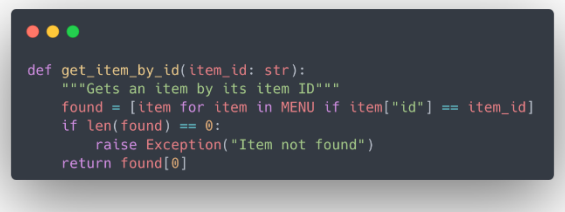
4. Legacy and Interactive User Interfaces

The interface provides:

- Interactive menu selection with icons
- Input validation and error handling
- Back navigation options
- Confirmation prompts for critical actions
- Switch between legacy and interactive UIs

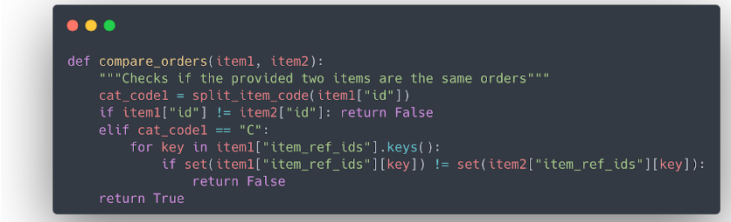
Functions used in this program

Dataset functions

Function Code	Description
	Returns a table like list of menu items for display purposes.
	Filters items from the menu by category code. Returns the list of items based on that category.
	Finds a specific item in the menu based on its unique ID. If the item doesn't exist, an exception is raised

	Splits an item ID into two separate parts: - Category code - Item no. Returns these two separate parts as a tuple.
	Finds specific items in the menu based on their IDs. If only category code is provided, the function will collect items based on that category. Returns a list of items found.

Utility functions

Function Code	Description
	Truncates long texts by returning a small part of the text with a '…'
	Compares two items in different parameters and decides if they are the same order or not.

```python
def parse_item_ref_ids(item_ref_ids: dict) -> list:
    """
    Parses the item_ref_ids dictionary from a combo meal.
    Returns a list of dictionaries with section as key and a dict as value:
    ```python
    {
        'section': str,
        'options': [...],
        'quantity': int,
        'locked': bool
        ...
    }
    ```
    - `'options'`: `list` of item IDs or category codes available for selection
    - `'quantity'`: how many can be selected from options
    - `'locked'`: `True` if `quantity == len(options)` (user cannot change selection)
    """
    parsed = []
    for section, (options, quantity) in item_ref_ids.items():
        if quantity > len(options):
            raise ValueError(f"Quantity for section '{section}' cannot be greater than number of options.")

        parsed_options = get_items_by_ids(options)
        parsed.append({
            'section': section,
            'options': parsed_options,
            'quantity': quantity,
            'locked': quantity == len(parsed_options)
        })
    return parsed
```

Handles combo meal structure by parsing item reference dictionaries and validating quantities against the available options

```python
def print_receipt(discount: float = 0.0):
    """Prints a receipt containing the items ordered (calculation will automatically be done)"""
    # Name too long, convert to smaller variables for cleaner code
    col_id = WIDTH_RECEIPT_TABLE_COLUMN_ID
    col_name = WIDTH_RECEIPT_TABLE_COLUMN_NAME
    col_type = WIDTH_RECEIPT_TABLE_COLUMN_TYPE
    col_price = WIDTH_RECEIPT_TABLE_COLUMN_PRICE
    col_qty = WIDTH_RECEIPT_TABLE_COLUMN_QUANTITY
    col_desc = WIDTH_RECEIPT_TABLE_COLUMN_DESCRIPTION

    # Set headers accordingly
    header_fields = (
        f"{'No.':<3} {'ID':<{col_id}} {'Name':<{col_name}} "
        f"{'Description':<{col_desc}} {'Type':>{col_type}} "
        f"{'Price':>{col_price}} {'Qty':>{col_qty}}"
    )

    # Calculations
    subtotal = sum(item["item_price"] * item["quantity"] for item in cart)
    discount_amt = round(subtotal * discount, 2)
    discounted_subtotal = subtotal - discount_amt
    gst = round(discounted_subtotal * GST, 2)
    total = round(discounted_subtotal + gst, 2)
    now = datetime.now().strftime("%d %b %Y   %H:%M")

    # Print Header
    print(f"\n{RESTAURANT_NAME:^{receipt_width}}")
    print(f"{ADDRESS:^{receipt_width}}")
    print(f"{'Tel: ' + PHONE_NUMBER:^{receipt_width}}")
    print(f"{WEBSITE:^{receipt_width}}")
    print("-" * receipt_width)
    print(f"Date: {now}")
    print("-" * receipt_width)

    # Table Body
    print(header_fields)
    print("-" * receipt_width)
    for i, item in enumerate(cart, 1):
        type_str = "Combo" if "options" in item else "À la carte"
        # Truncate name and description
        disp_name = condense(item.get('name', ''), col_name)
        disp_desc = condense(item.get('description', ''), col_desc)
        print(f"{i:<3} {item['id']:<{col_id}} {disp_name:<{col_name}} {disp_desc:<{col_desc}} {type_str:>{col_type}} "
              f"{item['item_price']:>{col_price}.2f} {item['quantity']:>{col_qty}}")

        # Options (for combos)
        if "options" in item:
            spaces = " " * (col_id + col_name + 6)
            for option_name, ids in item["options"].items():
                print(spaces + f"{condense(option_name, 12)}:")
                count = 1
                current_id = ""
                for item_id in ids:
                    # Check for similarity to display quantity
                    if current_id == item_id: count += 1
                    else:
                        count = 1
                        current_id = item_id
                    option_item_name = condense(str(count) + " " + get_item_by_id(item_id)['name'], col_name)
                    print(spaces + f"{option_item_name}")
    print("-" * receipt_width)

    # Display calculations
    print(f"{'Subtotal:':<{receipt_width - 10}}{subtotal:>10.2f}")
    if discount > 0.0:
        print(f"{f'Discount {discount*100:.0f}%:':<{receipt_width - 10}}-{discount_amt:>9.2f}")
    print(f"{f'GST {GST*100:.0f}%:':<{receipt_width - 10}}{gst:>10.2f}")
    print("-" * receipt_width)
    print(f"{'TOTAL:':<{receipt_width - 10}}{total:>10.2f}")
    print("-" * receipt_width)

    # Footer
    print()
    print(f"{'Thank you for dining with us!':^{receipt_width}}")
    print(f"{'We hope to see you again.':^{receipt_width}}\n")
```

Generates a comprehensive receipt with proper formatting, tax calculations, and discount applications

```python
def display_table(data: list[list[str]], headers: list[str] = [], selected_index: int = -1, tab_space: int = 4):
    """Prints a formatted table (compatible with interactive menu selection)"""
    # Calculate max width for each column
    col_widths = [max(len(str(cell)) for cell in col) for col in zip(*([headers] + data if headers else data))]
    # Helper function to format a row
    def fmt_row(row): return "|" + "|".join(f" {str(cell):<{w}} " for cell, w in zip(row, col_widths)) + "|"
    # Helper function to format a line
    def fmt_line(): return "+" + "+".join("-" * (w + 2) for w in col_widths) + "+"
    # If is interactive menu selection we give a little padding in order to show the >< cursor for selection
    # This is only if selected_index > 0
    padding = " " * tab_space if selected_index > -1 else ""
    if len(headers) > 0:
        print(padding, fmt_line())
        print(padding, fmt_row(headers))
    print(padding, fmt_line())
    # enumerate() gives an iteration with both indexes and the element for each element
    for i, row in enumerate(data):
        # Highlight selected row if needed
        if selected_index > -1 and i == selected_index: print(f"{'> ':>{tab_space}}", fmt_row(row), "<")
        else: print(padding, fmt_row(row))
    print(padding, fmt_line())
```

Provides an interactive table display with selection highlighting and customizable spacing.

Handler functions

```python
def handle_ui_integer_selection(
    question: str,
    allowed_min: int = -sys.maxsize,
    allowed_max: int = sys.maxsize,
    back_button: bool = False):
    """Handles integer selection UI"""
    while True:
        try:
            print(question)
            if back_button: print("To go back, type 'B'")
            value = input(f">> ")
            if back_button and value == "B":
                return None
            converted_value = int(value)
            if converted_value < allowed_min:
                print(f"❌ Value cannot be below {allowed_min}")
            elif converted_value > allowed_max:
                print(f"❌ Value cannot be above {allowed_max}")
            else:
                return converted_value
        except ValueError:
            print("❌ Value must be an integer")
```

```python
def handle_ui_menu_selection(
        question: str,
        options: list,
        option_icons: list = [],
        back_button: bool = False,
        confirm_button: bool = False,
        next_button: bool = False
    ):
    """Handles menu selection UI"""
    # Utility button icons
    back_icon = "⬅️ "
    confirm_icon = "✅ "
    next_icon = "➡️ "
    if not options:
        raise RuntimeError("Options cannot be empty")
    if option_icons and len(option_icons) != len(options):
        raise RuntimeError("Length of option_icons must match options")
    ui_icons = ((
        option_icons[:] if option_icons else [""] * len(options))
        + ([confirm_icon] if confirm_button else [])
        + ([next_icon] if next_button else [])
        + ([back_icon] if back_button else [])
    )
    while True:
        print(question)
        for i, (opt, icon) in enumerate(zip(options, ui_icons)):
            print(f"({i+1}) {icon + ' ' if icon else ''}{opt}")
        if confirm_button: print(f"(C) {confirm_icon} Confirm")
        if next_button: print(f"(N) {next_icon} Next")
        if back_button: print(f"(B) {back_icon} Back")
        sel = input(">> ")
        if sel == "C" and confirm_button: return "confirm"
        if sel == "N" and next_button: return "next"
        if sel == "B" and back_button: return "back"
        if sel.isdigit() and 1 <= int(sel) <= len(options): return int(sel) - 1
        print("❌ Invalid option!")
```

```python
def handle_edit_combo(item_id: str, preselected: dict = {}):
    """
    Handles editing of a combo meal by selecting items for each section.
    Optionally takes a `preselected` dict mapping section names to lists of selected option indices.
    """
    cat_code, _ = split_item_code(item_id)
    if cat_code != "C":
        raise ValueError("Only combo meals can be edited.")

    item_info = get_item_by_id(item_id)
    if not item_info:
        raise ValueError(f"Item with ID {item_id} not in Menu")

    # Get icon and parse combo sections
    parsed_sections = parse_item_ref_ids(item_info["item_ref_ids"])
    section_names = list(item_info["item_ref_ids"].keys())

    # Prepare initial selection state
    selected = []
    for i, section in enumerate(parsed_sections):
        sec_name = section_names[i]
        if section["locked"]:
            selected.append(list(range(len(section["options"]))))
        elif sec_name in preselected.keys():
            # Only accept valid items
            valid_indices = [i for i in range(len(section["options"])) if section["options"][i]["id"] in preselected[sec_name]]
            selected.append(valid_indices)
        else:
            selected.append([])

    # Legacy Menu (use input and handle_ui_menu_selection)
    result = preselected
    current_selection_index = 0
    back = False
    while current_selection_index < len(parsed_sections):
        if back:
            current_selection_index -= 1
            if current_selection_index < 0: return None
            back = False
            continue
        section = parsed_sections[current_selection_index]
        print("-" * 20)
        print(f"[ℹ️  Section {current_selection_index + 1}] {section["section"]}")
        if section.get("locked", False):
            # Locked section, just show the options
            print(f"🔒 This section is locked. Items included:")
            for option in section["options"]:
                item = get_item_by_id(option["id"])
                print(f" - {item['name']}")
            result[section["section"]] = [item["id"] for item in section["options"]]
            selected = handle_ui_menu_selection(
                "What do you want to do?",
                ["Next Section", "Previous Section"],
                ["➡️ ", "⬅️ "]
            )
            back = selected == 1
        else:
            chosen = []
            # Add saved data from result if any
            chosen += result.get(section["section"], [])

            # Do the selection part
            print(f"💡 You are allowed to select {section['quantity']} items for this section.")
            while True:
                option_names = []
                for option in section["options"]:
                    selected_mark = "✅ " if option["id"] in chosen else "⬛ "
                    option_names.append(f"{selected_mark} [{option["id"]}] {option['name']} (${option['price']:.2f})".strip())
                # Use handle_ui_menu_selection for selection, with back button
                sel = handle_ui_menu_selection(
                    f"Please select",
                    options=option_names,
                    back_button=True,
                    confirm_button=True
                )
                if sel == "confirm": break
                if sel == "back":
                    back = True
                    break
                # Find the index of the selected option
                selected_code = section["options"][sel]["id"]
                if selected_code in chosen:
                    chosen.remove(selected_code)
                else:
                    chosen.append(selected_code)
            if len(chosen) != section["quantity"] and not back:
                print(f"❌ You must select exactly {section['quantity']} item(s) for '{section["section"]}'")
                continue
            result[section["section"]] = chosen
        if not back: current_selection_index += 1
    return result
```

```python
def handle_checkout():
    """Handles the checkout menu system, exits after printing receipt"""
    if len(cart) == 0:
        print("❌ No items to pay. Please order an item before proceeding to checkout.")
        return None

    identity = handle_ui_menu_selection(
        "Please select your identity:",
        options=["Student", "Staff", "Loyalty Member", "I am not any one of these"],
        option_icons=["🎓 ", "💼 ", "💎 ", "❌ "],
        back_button=True
    )

    if identity == 0: discount_rate = DISCOUNT_RATES["student"]
    elif identity == 1: discount_rate = DISCOUNT_RATES["staff"]
    elif identity == 2: discount_rate = DISCOUNT_RATES["loyalty_member"]
    elif identity == 3: discount_rate = 0.0
    else: return None

    print("Printing receipt...")
    print_receipt(discount_rate)
    exit(0)
```

```python
def handle_edit_cart():
    """
    Handles viewing and editing the items inside the user cart
    """
    if len(cart) == 0:
        print("ℹ️ There are currently no items in your cart.")
        return

    table_headers = ["Index", "Code", "Category", "Name", "Price", "Quantity", "Total"]
    def show():
        """Output items in cart"""
        print(cart)
        display_table([
            [
                str(i + 1), cart[i]["id"],
                MENU_ITEM_IDS[split_item_code(cart[i]["id"])[0]][1], cart[i]["name"],
                f"${cart[i]["item_price"]:.2f}",
                cart[i]["quantity"],
                f"${cart[i]["item_price"] * cart[i]["quantity"]:.2f}"
            ]
            for i in range(len(cart))
        ], table_headers)
    while True:
        show()
        print("Select items by their indices separated by commas or B to go back.")
        inputs = input(">> ").split(",")
        if len(inputs) == 0:
            print("💡 To go back, type B and press enter.")
            continue
        elif inputs[0] == "B":
            return None

        while True:
            # Index validation
            indices = []
            for inp in inputs:
                if not inp.strip().isdigit():
                    print(f"> '{inp.strip()}' is not a valid index")
                    continue
                elif int(inp) - 1 < 0 or int(inp) - 1 >= len(inputs):
                    print(f"> '{inp.strip()}' is out of range.")
                    continue
                indices.append(int(inp) - 1)
            if len(indices) != len(inputs):
                error_count = len(inputs) - len(indices)
                if error_count == 1:
                    print(f"Please fix this error in your input.")
                else:
                    print(f"Please fix these {len(inputs) - len(indices)} errors in your input.")
                break


            action = handle_ui_menu_selection(
                "What would you like to do?",
                ["Change quantity", "Edit item", "Delete item(s)"],
                ["↔️ ", "✏️ ", "🗑️ "],
                back_button=True
            )
            if action not in ['C', 'E', 'D', 'B']: print("❌ Invalid option!")
            elif action == "C":
                new_quantity = handle_ui_integer_selection(
                    "Please type in the new quantity.",
                    allowed_min=1,
                    allowed_max=100,
                    back_button=True
                )
                if new_quantity:
                    for i in indices: cart[i]["quantity"] = new_quantity
                    break
            elif action == "E":
                if len(indices) > 1:
                    print("❌ You can only edit 1 item at a time.")
                elif split_item_code(cart[indices[0]]["id"])[0] != "C":
                    print("❌ Cannot edit à la carte items. You can only edit combo items.")
                else:
                    combo_selected_data = handle_edit_combo(cart[indices[0]]["id"], cart[indices[0]]["options"])
                    if combo_selected_data:
                        cart[indices[0]]["options"] = combo_selected_data
                        break
            elif action == "D":
                # Follow last index first method: Prevent errors from being raised when delete action is used
                # Sort out indices in descending order
                indices.sort(reverse=True)
                for i in indices: cart.pop(i)
                if len(cart) == 0:
                    print("ℹ️ There are currently no items in your cart.")
                    return None
                break
            elif action == "B": break
```

New functions used in our program:

Function Name	How it works
`zip(iter1, iter2, iter...)`	Combines two or more iterables into a single iterable for each element, which continues until the shortest iterable is exhausted
`enumerate(iter, start=0)`	Gives a count for each element inside an iterable
Note: Iterables are your strings, lists, tuples or anything that has elements which can be iterated with for and while loops.	

Conclusion

User Interface Design

The system implements a sophisticated interface with:

- **Menu Navigation:** Tab-based selection system for different categories
- **Item Selection:** Interactive highlighting with visual feedback
- **Input Validation:** Error handling for user inputs
- **Visual Enhancement:** Formatted tables for better user experience
- **UI Switching functionality:** Allows the switch between Legacy and Interactive UIs with just a single variable change

Advantages	Drawbacks
Interactive Terminal UI: Features like key navigation, live menu make user easier while ordering the meal.	**Code Complexity:** Some parts of our program logic may be hard to understand.
Item Selection by Category: Users can browse and order from structured menu categories (e.g., burgers, sides).	**No Data Persistence**: Our program does not store data if system is closed or restarted. Adding database integration and user authentication would allow order history tracking and inventory management.
Combo Meal Customization: Allows editing of combo items with dynamic option selection, supporting advanced order flexibility	**Error Messaging and Flow Control**: We have minimized the need to always reorder in our program. However, there will be parts where it may cause users to repeat their orders, which could be frustrating in real-time usage.

Thank you!