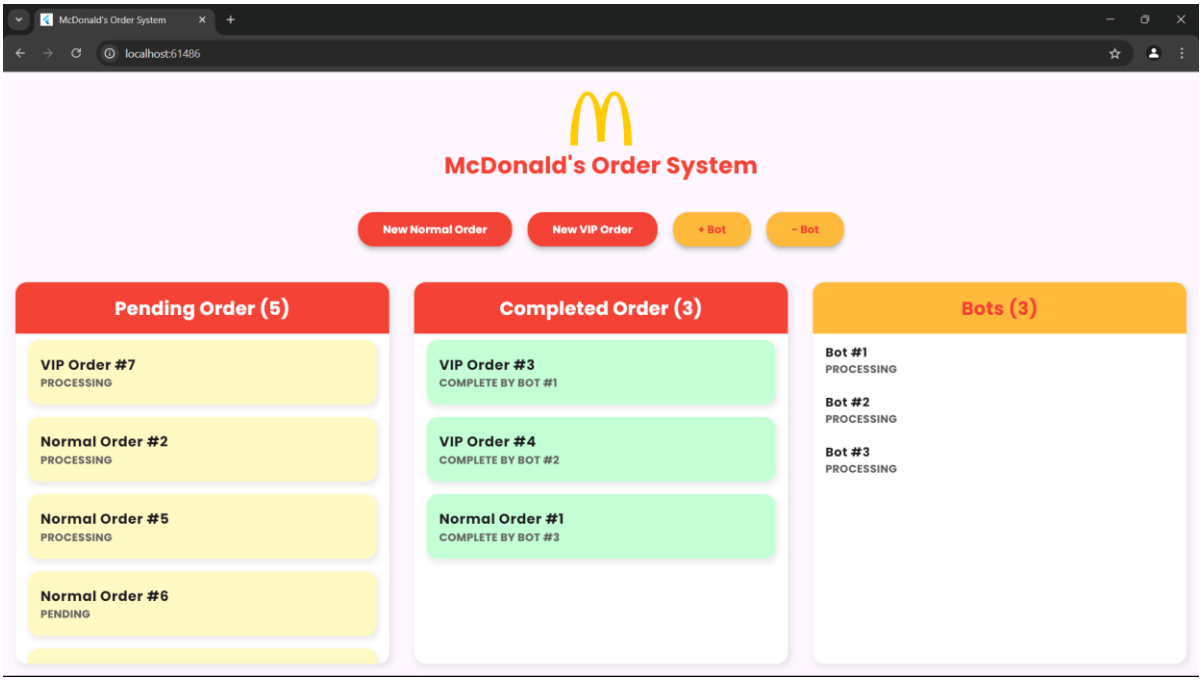


FeedMe Software Engineer Take Home Assignment: McDonald's Order Controller System

Lead Programmer & UI/UX Designer: Amirah Zulkifli

Video Demo: https://youtu.be/IgMIy34O7PE

Programming Languages/ FrameWork/ Tools: Dart, Flutter & Visual Studio Code

Functions	Explanations

`void addOrder(String type) {`
`  setState(fn: () { // rebuild the UI with updated state`
`  final Order newOrder = Order(number: orderNumber++, type: type, status: 'PENDING');`
`  if (type == 'VIP') {`
`  int vipIndex = pendingOrders.indexWhere(test: (Order order) => order.type != 'VIP');`
`  if (vipIndex == -1) { //if all orders in the list are vip/empty`
`  pendingOrders.add(value: newOrder); //added to the end of the list for the vips`
`  } else {`
`  pendingOrders.insert(index: vipIndex, element: newOrder); //inserted at the positio`
`  }`
`  } else {`
`  pendingOrders.add(value: newOrder); //add normal to pendinglist`
`  }`
`  processOrders(); //asign pending orders to available bots`
`  });`
`}`
	This function adds a new order to the pending order list with the type either "VIP" or "Normal" and assign it unique order number with a status of 'PENDING'. It also ensures that VIP orders are prioritized over normal orders where the VIP order is inserted before the first normal order.

`void addBot() {`
`  setState(fn: () {`
`  final Bot bot = Bot(id: ++botCount, status: 'IDLE'); //`
`  bots.add(value: bot); //add the new bot to the list`
`  processOrders(); //assign pending orders to available`
`  });`
`}`
	This function creates a new bot with a unique id and sets its status to 'IDLE', adds it to the list of bots, and then proceed to assign pending orders to idle bots to process the order.

```
void processOrders() async {
  for (var Bot bot in bots) { //lop through bots and assign orders to available bots
    if (bot.status == 'IDLE' && pendingOrders.isNotEmpty) {
      final Order order = pendingOrders.firstWhere(test: (Order order) => order.status == 'PENDING');

      setState(fn: () {
        order.status = 'PROCESSING';
        bot.status = 'PROCESSING';
        bot.currentOrder = order; // Assign the order to the bot
        order.botId = bot.id; //asign the order's bot id
      });

      bot.timer = Timer(duration: Duration(seconds: 10), callback: () { //sets up a 10-second timer
        setState(fn: () {
          pendingOrders.remove(value: order); //remove the order from the pending order
          completedOrders.add(value: order..status = 'COMPLETE'); //add the order to the complete or
          bot.status = 'IDLE';
          bot.currentOrder = null; // Clear the order after processing
          processOrders();
        });
      }); // Timer
    }
  }
}
```

This functions processes orders by assigning any pending order to any idle bots. Each bot takes 10 seconds to process one order. The function will loop through each bot in the bots list and check if a bot's status is 'IDLE' and there are pending orders, it will pick the first pending order to process and eventually change the order's and bot's status to 'PROCESSING'. Once the timer finishes, the pending order is removed from the pending orders list and added to the completed order list with the status updated to 'COMPLETE'. The bot's status is reset to 'IDLE'. The functions then calls processOrders() again to check if any more pending orders can be processed.

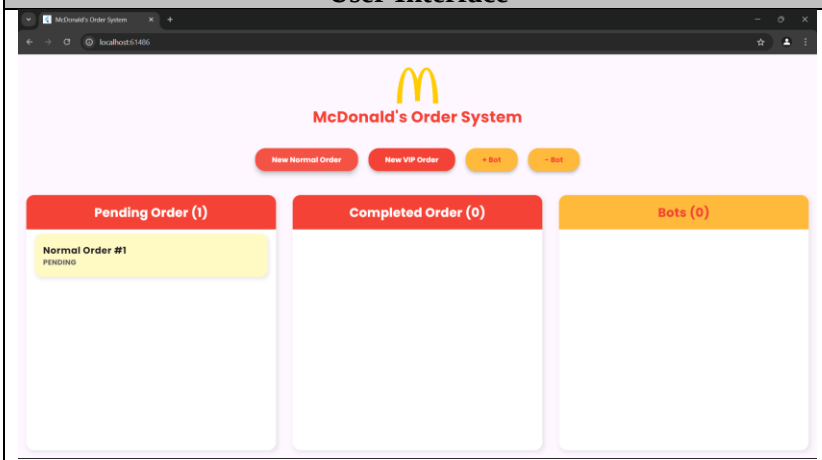
```
void removeBot() {
  if (bots.isNotEmpty) {
    setState(fn: () {
      Bot lastBot = bots.removeLast();

      // Check if the bot is currently processing an order
      if (lastBot.status == 'PROCESSING' && lastBot.currentOrder != null) {
        // Cancel the timer for the bot
        lastBot.timer?.cancel();
        lastBot.currentOrder!.status = 'PENDING';  // Set the order status ba
        lastBot.currentOrder = null; // Clear the bot's current order
        processOrders();  // Reassign the pending orders to available bots
      }
    });
  }
}
```

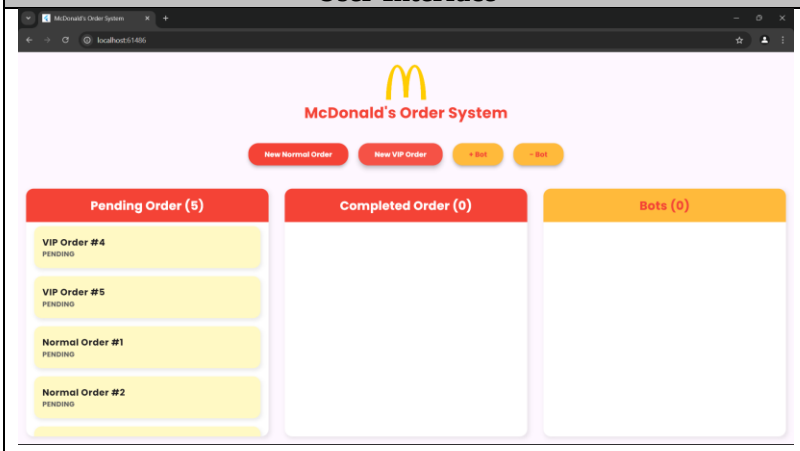
This function removes the last bot from the list of bots. If the removed bots is in the middle of processing an order, the bot's timer is cancelled, and the order is assigned to the pending list. The processOrder() function is called again to assign pending orders to available bots.

Requirements:

1. When "New Normal Order" clicked, a new order should show up "PENDING" Area.

User Interface	Explanation
	When the user clicks the "New Normal Order" button, it triggers the creation of a new order by passing the order type ("Normal" in this case) to the addOrder() function. Inside this function, a new order is generated with a unique order number, the specified type, and its status set to "PENDING." This new order is then added to the pendingOrders list, which causes it to appear in the "PENDING" section of the UI.

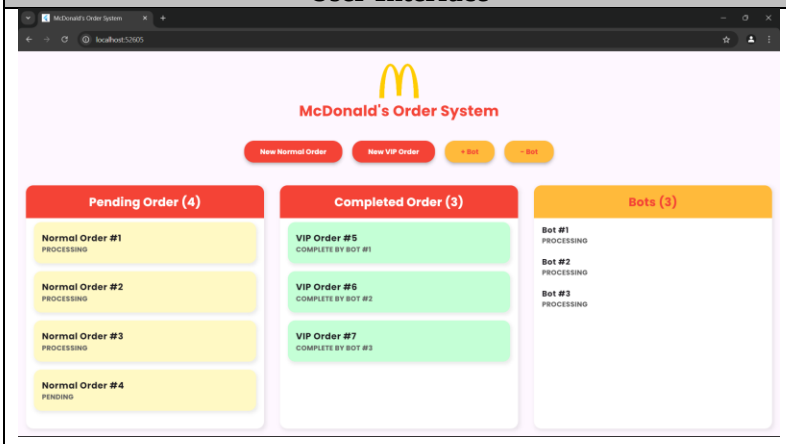
2. When "New VIP Order" clicked, a new order should show up in "PENDING" Area. It should place in-front of all existing "Normal" order but behind of all existing "VIP" order.

User Interface	Explanation
	When the user clicks the "New VIP Order" button, the addOrder() function checks if the order type is "VIP." If it is, the function searches through the pendingOrders list to find the position of the first "Normal" order. The new VIP order is then inserted at this position, ensuring that it appears in front of all existing "Normal" orders but behind any existing "VIP" orders. This maintains the priority order, with VIP orders being processed ahead of Normal orders.

3. The order number should be unique and increasing.

   Explanation: Every new order, whether "Normal" or "VIP," is assigned a unique, incrementing order number starting from 1. The order number increases sequentially each time a new order is added, ensuring no two orders share the same number.

4. When "+ Bot" clicked, a bot should be created and start processing the order inside "PENDING" area. After 10 seconds picking up the order, the order should move to "COMPLETE" area. Then the bot should start processing another order if there is any left in "PENDING" area.
5. If there is no more order in the "PENDING" area, the bot should become IDLE until a new order come in.

User Interface	Explanation
	When the "+ Bot" button is clicked, it triggers the addBot() function to create a new bot. Initially, the bot's status is set to "IDLE." If there are pending orders, the bot starts processing them using the processOrders() function. The first pending order's status is changed to "PROCESSING," and the bot is assigned to it. After 10 seconds, the bot completes the order. The order is removed from the pendingOrders list, added to the completedOrders list, and its status is updated to "COMPLETE." The bot then returns to "IDLE" and checks for other pending orders. If there are

	more pending orders, the bot automatically begins processing the next one. If no pending orders remain, the bot stays "IDLE" until a new order is added.

6. When "- Bot" clicked, the newest bot should be destroyed. If the bot is processing an order, it should also stop the process. The order now back to "PENDING" and ready to process by other bot.

   Explanation: When the “-Bot” is clicked, it triggers the removeBot() function to remove the most recently added bot. If the bot being removed is currently idle, it is simply removed from the bot list. If the bot is in the middle of processing an order, the bot’s timer is canceled, and the order that was being processed is returned to the "PENDING" state. The order that was interrupted is now ready to be picked up by another available bot. The system reassigns it to any idle bots through the processOrders() function.