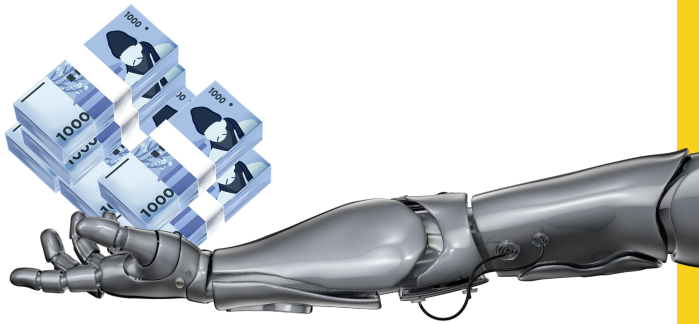
천원의 아침밥 식당 로봇 서빙 시스템 구축

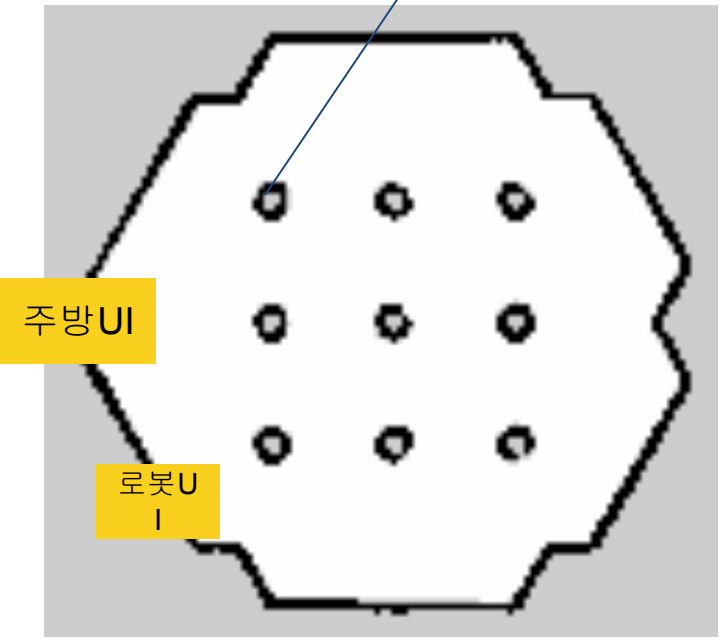
C-2



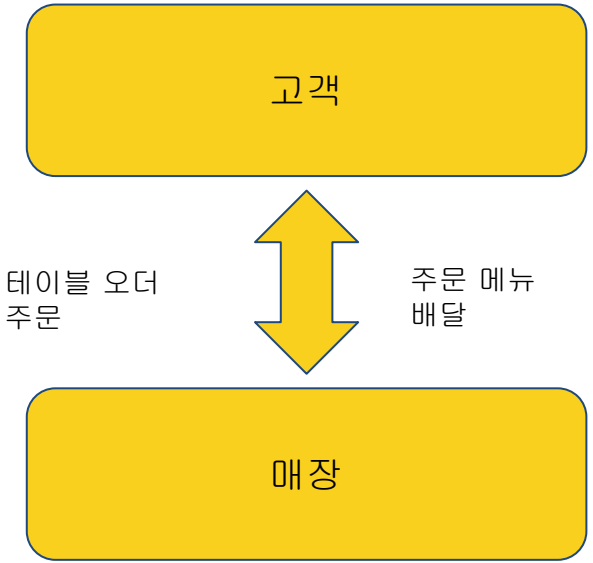
목차
Index

시나리오

1. Situation



2.Business



시나리오 요구조건

테이블 오더	• 수량 조절 가능 • DB에 주문정보 저장 • 주간 메뉴판 디스플레이 • 화면에 총 금액, 주문완료 표시
주방 디스플레이	• 테이블 별 주문 확인가능 • MainWindow에 잔여 배급 가능 수량 표시 +매니저용 디스플레이: 일 주문 비율 최근 7일간 주문수량 추이 등 데이터 계산
서빙 로봇	• 대기 중, 배달 중, 복귀 중 3가지 디스플레이 구현, • 각 로봇 상태에 적절한 캐릭터 gif 파일 사용으로 고객에게 친숙함 • 손님과의 상호작용: 수령완료 버튼

DB구조1_특정 주기 마다 Update

1. menu - 확장성 있게 제작

id	menu	price
1	급식	1,000
2	빵식	1,000

2. daiy_menu - 최근 한달 동안의 식사 리스트

id	day	menu_id	comp_1	comp_2	comp_3	comp_4
1	2025-01-01 00:00:00	1	흰밥	순두부찌개	삼겹살구이	무나물
2	2025-01-01 00:00:00	2	식빵	스크램블에그	베이컨	치즈
3	2025-01-02 00:00:00	1	잡곡밥	김치찌개	닭볶음탕	콩나물무침
4	2025-01-02 00:00:00	2	크로와상	감자샐러드	햄	버터
5	2025-01-03 00:00:00	1	현미밥	미역국	제육볶음	시금치나물
6	2025-01-03 00:00:00	2	모닝빵	감자크로켓	치킨텐더	머스타드
7	2025-01-04 00:00:00	1	쌀밥	된장찌개	불고기	나물
8	2025-01-04 00:00:00	2	바게트	햄치즈샌드위치	토마토	버터
9	2025-01-05 00:00:00	1	찹쌀밥	육개장	고등어구이	감자조림
10	2025-01-05 00:00:00	2	식빵	스크램블에그	베이컨	치즈
11	2025-01-06 00:00:00	1	흰밥	순두부찌개	삼겹살구이	무나물
12	2025-01-06 00:00:00	2	식빵	스크램블에그	베이컨	치즈
13	2025-01-07 00:00:00	1	쌀밥	된장찌개	불고기	나물
14	2025-01-07 00:00:00	2	모닝빵	감자크로켓	치킨텐더	머스타드
15	2025-01-08 00:00:00	1	흰밥	순두부찌개	삼겹살구이	무나물

3. supply - 식사 공급량

id	day	menu_id	remaining
1	2025-01-01 00:00:00	1	197
2	2025-01-01 00:00:00	2	84
3	2025-01-02 00:00:00	1	202
4	2025-01-02 00:00:00	2	90
5	2025-01-03 00:00:00	1	211
6	2025-01-03 00:00:00	2	75
7	2025-01-04 00:00:00	1	219
8	2025-01-04 00:00:00	2	79
9	2025-01-05 00:00:00	1	227
10	2025-01-05 00:00:00	2	81
11	2025-01-06 00:00:00	1	220
12	2025-01-06 00:00:00	2	88
13	2025-01-07 00:00:00	1	197

DB구조2_주문 마다 Update

4. orders - Table_order에서 주문시 기록

id	order_time	table_num	menu_id	num
11,250,131,063,632	2025-01-31 06:36:32	1	1	4
12,250,131,063,632	2025-01-31 06:36:32	1	2	2
21,250,131,063,629	2025-01-31 06:36:29	2	1	6
21,250,131,063,719	2025-01-31 06:37:19	2	1	6
22,250,131,063,629	2025-01-31 06:36:29	2	2	3
22,250,131,063,719	2025-01-31 06:37:19	2	2	3

5. order_confirm - kitchen에서 서빙 시작시 기록

id	time	processing	etc
11,250,131,063,632	2025-01-31 06:36:37	1	
12,250,131,063,632	2025-01-31 06:36:37	1	
21,250,131,063,629	2025-01-31 06:36:37	1	
21,250,131,063,719	2025-01-31 06:37:22	1	asdasdasd

DB

관리자 UI 객체 설명

1. 날짜 선택
2. 1-일일 메뉴별 주문수, 매출
3. 전체 주문비율
4. 평균 조리시간
5. 최신 주간 매출 추이
6. 시간대별 주문 수 :
7. 식품 발주 사이트 바로가기 url

시스템 설계 노드 구성

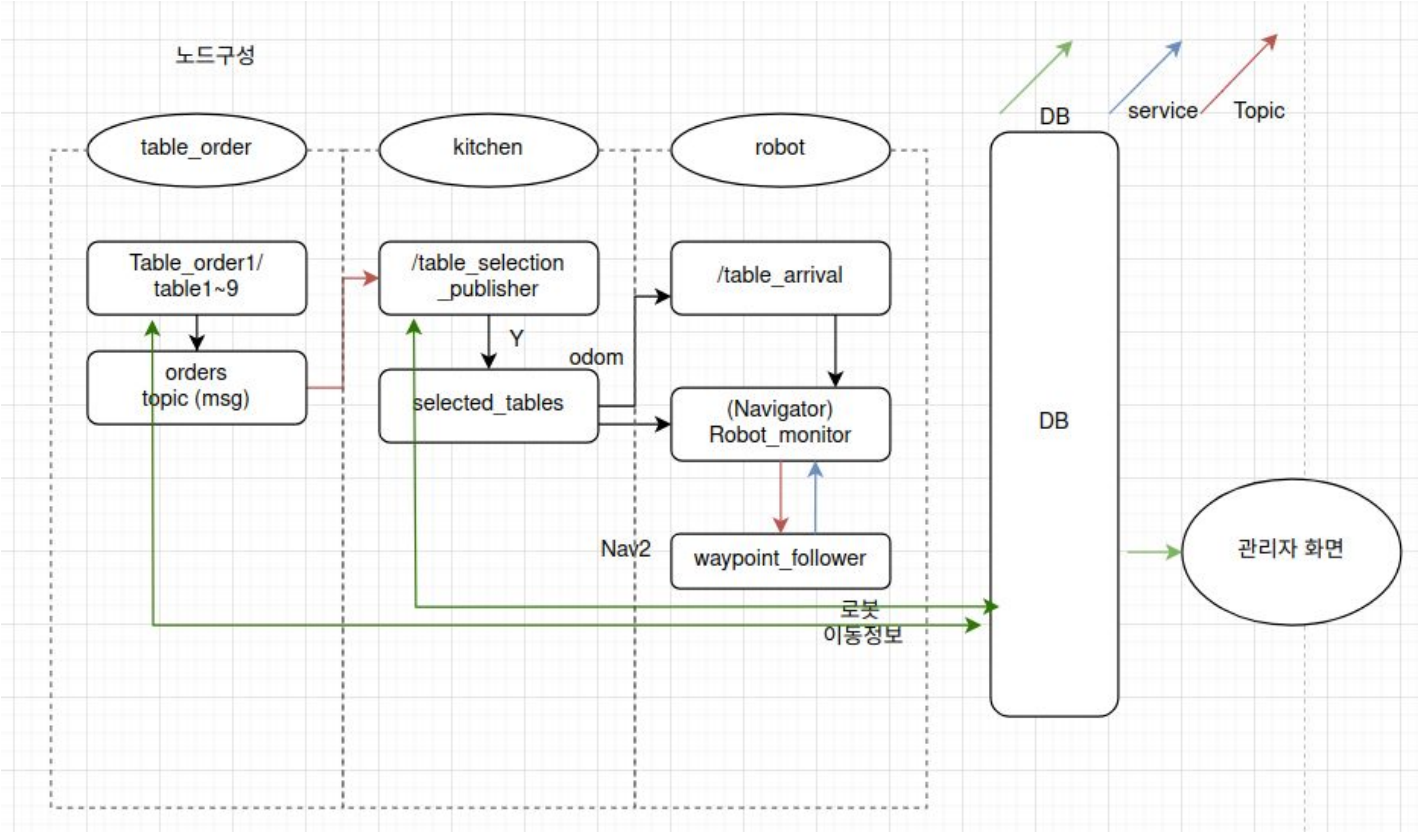

노드구성
table_order
kitchen
robot
DB
service
Topic
Table_order1/
table1~9
/table_selection
_publisher
/table_arrival
Y
odom
orders
topic (msg)
selected_tables
(Navigator)
Robot_monitor
DB
Nav2
waypoint_follower
관리자 화면
로봇
이동정보

Table order


/Table_order1
/Table_order2
: 천원의 아침밥 식당의 주간 메뉴를 확인하세요
주간 빵식메뉴
주간 급식메뉴
급식 : 1000원
현미밥, 미역국,
제육볶음,
시금치나물,
배추김치
4
추가
주문 확인
Food: 4
Bread: 3
빵식 : 1000원
바게트,
햄치즈샌드위치,
토마토, 버터,
초코잼
3
추가
주문중
Reset
OK


주간 빵식 메뉴
천원의 아침밥 식당의 주간 빵식 메뉴를 확인하세요!
2025-01-23
2025-01-24
2025-01-25
2025-01-26
2025-01-27
2025-01-28
2025-01-29
식빵
스크램블에그
베이컨
치즈
딸기잼
식빵
스크램블에그
베이컨
치즈
딸기잼
바게트
햄치즈샌드...
토마토
버터
초코잼
모닝빵
감자크로켓
치킨텐더
머스타드
포도잼
호밀빵
그린샐러드
훈제연어
크림치즈
오렌지잼
바게트
햄치즈샌드...
토마토
버터
초코잼
모닝빵
감자크로켓
치킨텐더
머스타드
포도잼

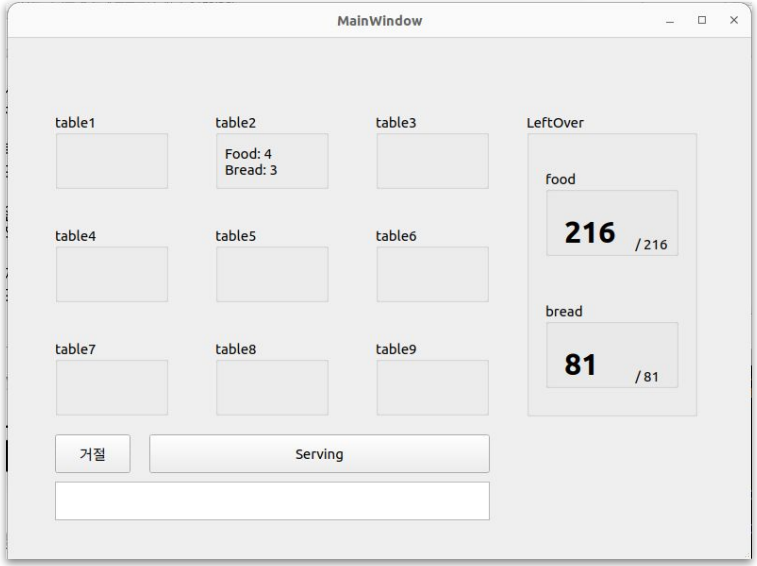
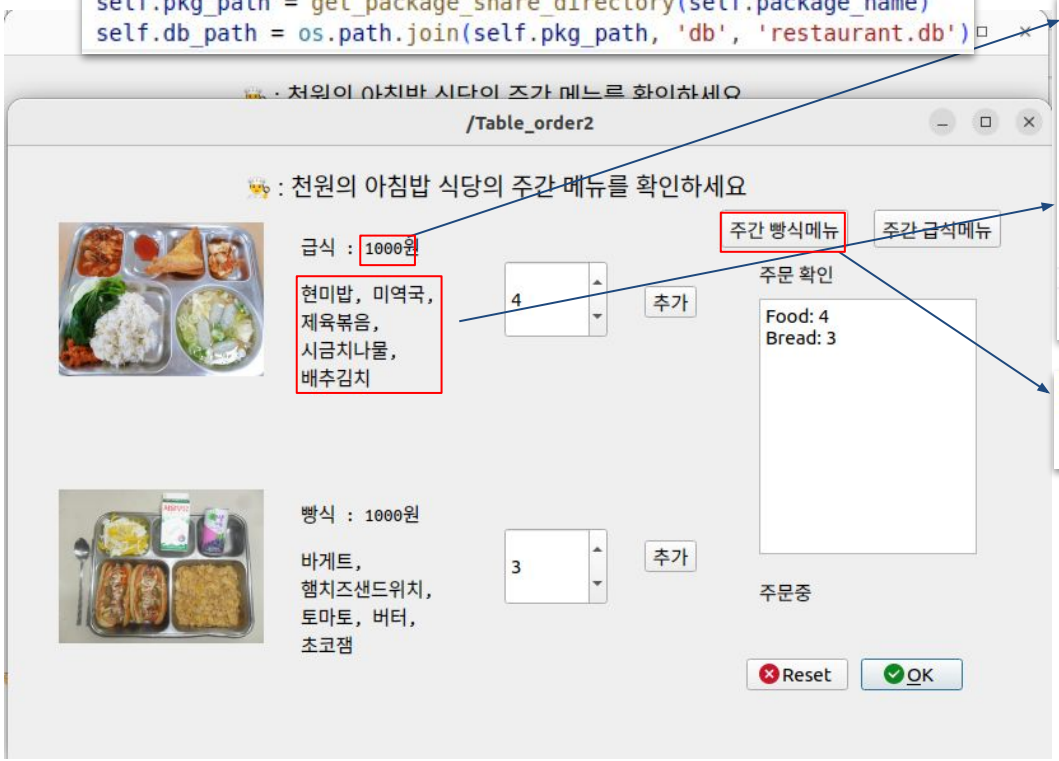

MainWindow
table1
table2
Food: 4
Bread: 3
table3
LeftOver
food
216
/ 216
table4
table5
table6
bread
81
/ 81
table7
table8
table9
거절
Serving

Table Order

```
self.package_name = 'serving_project'
self.pkg_path = get_package_share_directory(self.package_name)
self.db_path = os.path.join(self.pkg_path, 'db', 'restaurant.db')
```



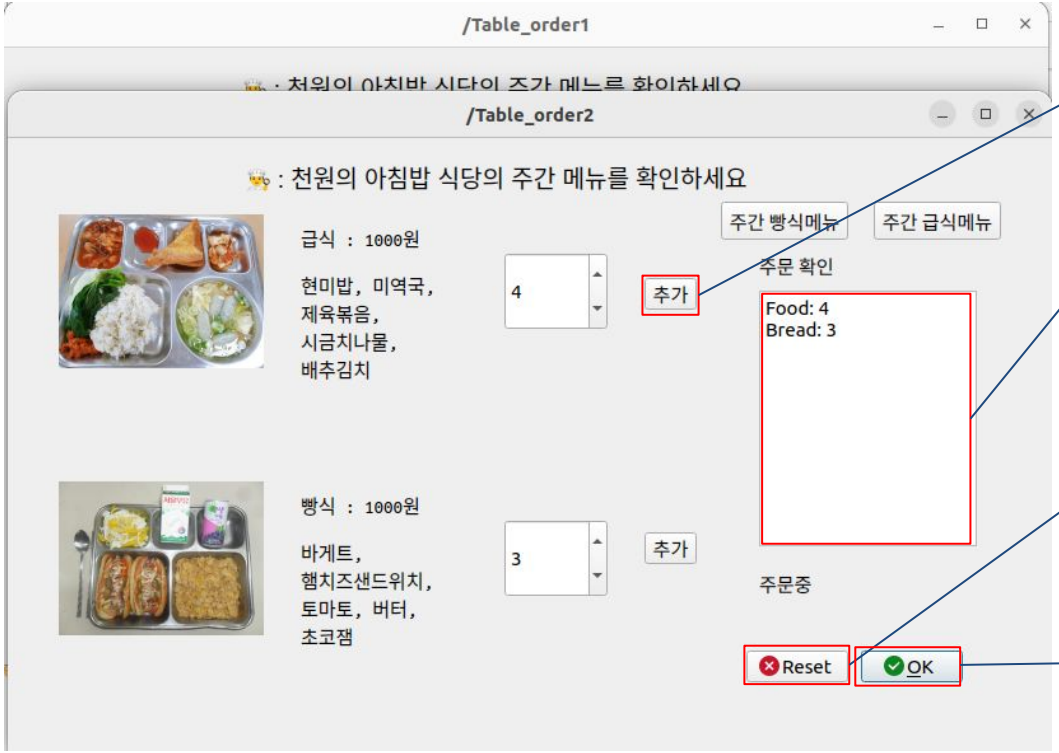

```
food_price = """
    SELECT price FROM menu
    WHERE id = 1;
"""
bread_price = """
    SELECT price FROM menu
    WHERE id = 2;
"""
food_query = """
    SELECT comp_1, comp_2, comp_3, comp_4, comp_5 FROM daily_menu
    WHERE DATE(day) = '2025-01-31' AND menu_id = 1;
"""
bread_query = """
    SELECT comp_1, comp_2, comp_3, comp_4, comp_5 FROM daily_menu
    WHERE DATE(day) = '2025-01-31' AND menu_id = 2;
```

Today

```
def show_weekly_menu(self):
    self.today_window = TodayWindow()
    self.today_window.show()
```

```
query = """
SELECT day, comp_1, comp_2, comp_3, comp_4, comp_5
FROM daily_menu
WHERE day BETWEEN DATE('now', '-7 days') AND DATE('now')
AND menu_id = 2
"""
```

Table Order




```python
def add_food_to_list(self):
    quantity = self.food_spinbox.value()
    self.food_num_total += quantity
    self.update_pick_list()
```

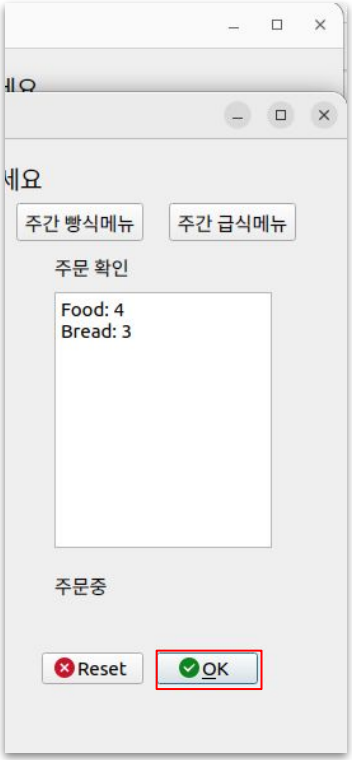
```python
def update_pick_list(self):
    self.pick_list.clear()
    if self.food_num_total > 0:
        self.pick_list.appendPlainText(f"Food: {self.food_num_total}")
    if self.bread_num_total > 0:
        self.pick_list.appendPlainText(f"Bread: {self.bread_num_total}")
```

```python
def reset_pick_list(self):
    self.pick_list.clear()
    self.food_num_total = 0
    self.bread_num_total = 0
```

```python
    self.publisher = self.create_publisher(String, 'orders', qos_profile)
    self.timer = self.create_timer(0.1, self.publish_order)
    self.get_logger().info("[INFO] ROS2Node initialized and publisher created.")

def publish_order(self):
    while not self.queue.empty():
        msg = self.queue.get()
        self.publisher.publish(msg)
        self.get_logger().info(f"[INFO] Published order message: {msg.data}")
```

Table Order



Ok 버튼을 눌렀을 때, 입력된 데이터를 SQLite DB에 저장하고, ROS2로 퍼블리시

```
"INSERT INTO orders (id, order_time, table_num, menu_id, num) VALUES (?, ?, ?, ?, ?)",
(food_order_id, order_time, table_num, 1, self.food_num_total)
```

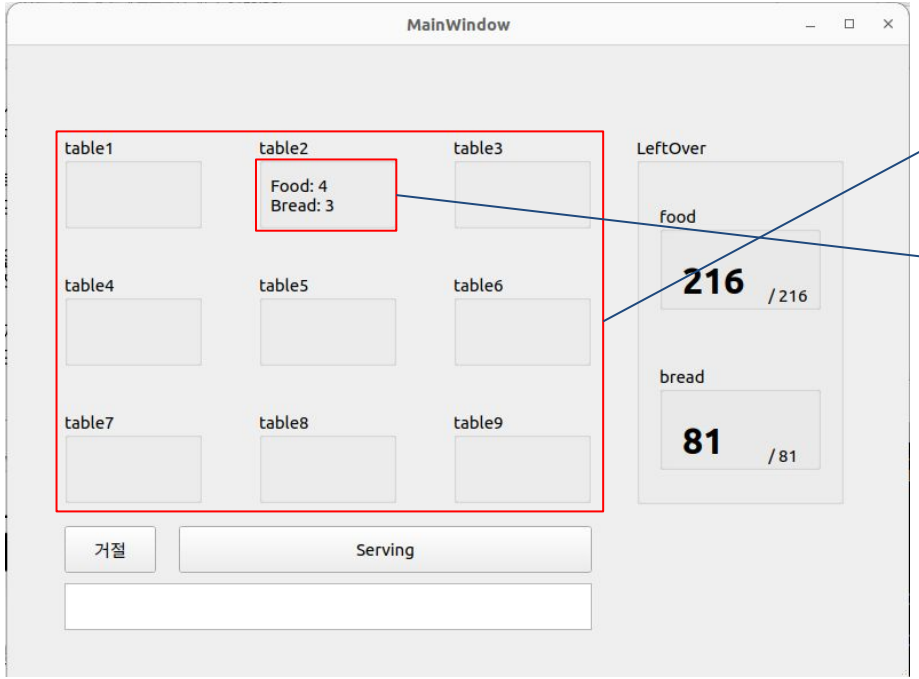
DB저장 및 topic 정보

food_order_id : table_num + food_id + current_time
order_time : 현재 시간
table_num : (thread)주문 테이블의 번호
menu_id : 음식 id
food_num_total : 총 주문 음식/빵식 수

```
# ROS2 메시지 생성 및 발행
msg = String()
msg.data = f"{bread_order_id}: Bread: {self.bread_num_total}"
self.node.queue.put(msg)
```

```
yigd:~/Documents/serving_project$ ros2 topic list
/Table_order1/orders
/Table_order2/orders
/Table_order3/orders
/Table_order4/orders
/Table_order5/orders
/Table_order6/orders
/Table_order7/orders
/Table_order8/orders
/Table_order9/orders
```

Kitchen




```
yigd:~/Documents/serving_project$ ros2 topic list
/Table_order1/orders
/Table_order2/orders
/Table_order3/orders
/Table_order4/orders
/Table_order5/orders
/Table_order6/orders
/Table_order7/orders
/Table_order8/orders
/Table_order9/orders
```

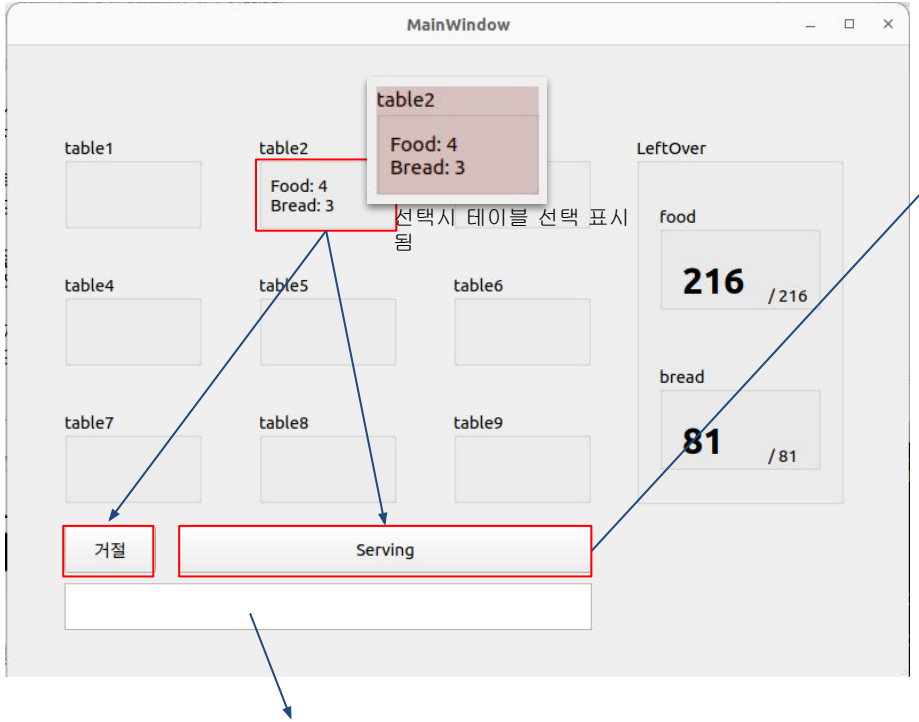
```
[k4-1] [INFO] [1738266677.906012260] [table_selection_publisher]
: Received message for table 2: 21250131045117: Food: 4
[k4-1] [INFO] [1738266677.906959273] [table_selection_publisher]
: Received message for table 2: 22250131045117: Bread: 3
```

/orders의 토픽 형태

각각의 테이블 마다 message형태로
table num, id, food_num 을 받아서
해당 테이블에 Food : n, Bread : m 정보만 보여줌

따로 주방(kitchen) 노드가 주문을 수락/거절하는 피드백은 안함

Kitchen



선택시 테이블 선택 표시 됨

table 한개만 선택시에 해당 건의 사항 및 문제점 제출 가능

1. ROS2에 선택된 테이블 정보 전송 (토픽 발행)

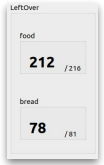
```
[k4-1] [INFO] [1738268469.320852062] [table_selection_publisher]
 Published: array('i', [2, 6, 8])
```

2. 서빙할 메뉴 개수 계산

```
for msg_data in orders:  # 주문 데이터 읽기
    food_match = re.search(r'Food:\s*(\d+)', msg_data)
    bread_match = re.search(r'Bread:\s*(\d+)', msg_data)
```

3. 서빙 후 남은 재고(supply) 업데이트

```
self.db_manager.update_remaining_count(1, food_total)
self.ui.label.setText(str(self.db_manager.initial_food_remaining))
```



4. 주문 완료 상태를 DB에 저장

```
self.ros2_publisher.insert_order_to_db(table_number, 'y', etc_text)
```

거절이면 n

5. 선택된(빨갛게) 된 부분 초기화

토픽

퍼블리시 (Publish)

토픽명	메시지 타입	퍼블리셔	내용
`orders`	`std_msgs/msg/String`	`ROS2Node` (`Ui_Dialog`)	사용자가 주문하면 주문 정보를 퍼블리시
`selected_tables`	`std_msgs/msg/Int32MultiArray`	`ROS2Publisher` (`Ui_MainWindow`)	주문 UI에서 서빙할 테이블 번호를 퍼블리시
`/table_arrival`	`std_msgs/msg/Int32`	`WaypointFollower` (로봇)	로봇이 테이블 도착 시 해당 테이블 번호를 퍼블리시

구독 (Subscribe)

토픽명	메시지 타입	서브스크라이버	내용
`/Table_order{i}/orders` (i=1~9)	`std_msgs/msg/String`	`ROS2Publisher` (`Ui_MainWindow`)	테이블별 주문을 구독하여 서빙 UI에서 관리
`selected_tables`	`std_msgs/msg/Int32MultiArray`	`WaypointFollower` (로봇)	주문 UI에서 서빙할 테이블 번호를 받아 로봇이 이동
`/table_arrival`	`std_msgs/msg/Int32`	`ROS2Publisher` (`Ui_MainWindow`)	로봇이 테이블 도착 시 UI에서 처리 가능

Table Order- log및 QOS설정

로그레벨	로그 메시지
INFO	Food Order Inserted (음식 주문 저장됨)
	Message sent to ROS2 (ROS2에 음식 주문 전송됨)
	Bread Order Inserted (빵 주문 저장됨)
	Message sent to ROS2 (ROS2에 빵 주문 전송됨)
	Order sent and label updated to '주문 완료' (UI 상태 변경됨)
	Published order message (ROS2 퍼블리싱됨)
	ROS2Node initialized and publisher created (ROS2 노드 초기화 완료)
WARNING	Order is empty, skipping send_order (주문 데이터가 없음)
ERROR	Database error while inserting order (DB 저장 실패)

QOS

```
qos_profile = QoSProfile(
    reliability=ReliabilityPolicy.RELIABLE,  # 주문은 반드시 전달되어야 하므로 신뢰성 보장
    history=HistoryPolicy.KEEP_LAST,  # 최신 주문만 유지
    depth=1  # 최신 1개 주문만 유지
)
self.publisher = self.create_publisher(String, 'orders', qos_profile)
```

주문은 **반드시 전달되어야 하므로 RELIABLE 설정**,

최신 주문만 유지(KEEP_LAST(1))

Kitchen - log및 QOS설정

로그레벨	로그 메시지
INFO	Updated table_orders[1]: ['11250131062252: Food: 3', '12250131062252: Bread: 5'] - 테이블 1번의 주문 목록이 정상적으로 업데이트됨
	서빙할 테이블 번호: [1]
	Published: array('i', [1]) - ROS2 메시지가 정상적으로 퍼블리시됨
	DB Insert Attempt - table_orders[1]: [] - DB 저장 전에 table_orders[1]이 비어 있음
WARNING	No orders found for table 1 - 주문이 없어서 DB에 저장할 수 없음
	imsi_t_ValueError: Cannot convert order_id to int - 22250131063308 - order_id 정수 변환 실패 (msg.data 처리 문제 가능성)
ERROR	Database error: datatype mismatch - DB 저장 시 데이터 타입 불일치 (processing 값 문제 가능성)
	ValueError: Cannot convert order_id to int - 22250131063308. Error: list.remove(x): x not in list - order_id가 리스트에 없는데 remove()를 시도함

ROS 설정

주문 수신(`orders`): 신뢰성 보장(RELIABLE), 과거 데이터 필요 없음(VOLATILE)

서빙 퍼블리시(`selected_tables`): 주문 처리 정보를 확실하게 전달(RELIABLE)

```
qos_profile = QoSProfile(
    reliability=ReliabilityPolicy.RELIABLE,  # 서빙 정보가 반드시 전달되어야 함
    depth=10  # 큐에 10개까지 유지
)

self.publisher_ = self.create_publisher(Int32MultiArray, 'selected_tables', qos_profile)
```

```
qos_profile = QoSProfile(
    reliability=ReliabilityPolicy.RELIABLE,  # 신뢰성 보장 (주문이 유실되면 안 됨)
    durability=DurabilityPolicy.VOLATILE,  # 과거 메시지는 중요하지 않음
    depth=10  # 최근 10개까지 유지
)

subscriber = self.create_subscription(
    String,
    topic_name,
    partial(self.listener_callback, table_number=i),
    qos_profile
)
```

서빙 로봇

```python
import rclpy  # ROS 2의 Python API를 가져옴
from rclpy.action import ActionClient  # 액션 클라이언트를 사용하기 위한 모듈
from rclpy.node import Node  # ROS 2 노드를 생성하기 위한 기본 클래스
from rclpy.qos import QoSProfile, QoSReliabilityPolicy  # ROS 2 메시지 QoS 설정을 위한 모듈
from std_msgs.msg import Int32MultiArray  # 정수 배열 데이터를 처리하는 메시지 타입
from geometry_msgs.msg import PoseStamped  # 위치(Pose) 데이터를 다루는 메시지 타입
from nav2_msgs.action import FollowWaypoints  # Nav2의 웨이포인트 추적 액션 메시지
import threading  # 키보드 입력 처리를 위한 별도 스레드 생성
import sys  # 시스템 관련 기능을 제공하는 모듈
import select  # 키보드 입력 감지를 위한 모듈
import termios  # 터미널 설정을 위한 모듈
import tty  # 터미널 입력 모드를 제어하기 위한 모듈
```

kitchen의 display와 소통하기 위한 필요 모듈

순서
주방에서 조리 완료 -> 조리 완성 순서대로 3개의 테이블에 배달->서빙 완료 후 테이블 오더와 부엌 디스플레이에 전달

이동순서
초기 좌표 -> 목표1 -> waypoint1->목표2 -> waypoint2 -> 목표3->waypoint3->초기 좌표

서빙 로봇_log

```python
class WaypointFollower(Node):
    def __init__(self):
        # 이동 상태 플래그
        self.is_moving = False

        # 새로운 명령을 저장하는 큐
        self.pending_tasks = []

        # 🚀 `selected_tables` 토픽 구독 (기존 `table_selection` → `selected_tables`)
        qos_profile = QoSProfile(depth=10, reliability=QoSReliabilityPolicy.RELIABLE)
        self.subscription = self.create_subscription(
            Int32MultiArray, '/selected_tables', self.table_selection_callback, qos_profile
        )

        self.get_logger().info("Subscribed to /selected_tables")

    def table_selection_callback(self, msg):
        """
        테이블 선택 토픽 수신 콜백 함수 (1~9개의 랜덤 테이블이 들어올 수 있음)
        """
        table_numbers = list(msg.data)
        self.get_logger().info(f'Received Table Numbers: {table_numbers}')

        if self.is_moving:
            # 🚀 현재 이동 중이면 대기열에 저장 후, 기존 경로 종료 후 실행
            self.pending_tasks.append(table_numbers)
            self.get_logger().info(f'Queued Task: {table_numbers}')
```

self.get_logger().info("Subscribed to /selected_tables")
:`/selected_tables` 토픽을 성공적으로 받았는지를 체크할 수 있습니다

self.get_logger().info(f'Received Table Numbers: {table_numbers}')
:사용자가 입력한 테이블 번호를 기록합니다. 이를 통해 잘못 기록 된 것이 있는지 확인합니다

self.get_logger().info("WaypointFollower services initialized!")
;주요 서비스 (`get_robot_position`, `get_robot_status`, `cancel_robot_movement`) 가 정상적으로 실행되었는지 확인합니다

self.get_logger().info('Goal accepted :)')
:로봇이 경로 이동을 시작했음을 알려줍니다

서빙 로봇_qos

```python
# 🚀 `selected_tables` 토픽 구독 (기존 `table_selection` → `selected_tables`)
qos_profile = QoSProfile(depth=10, reliability=QoSReliabilityPolicy.RELIABLE)
self.subscription = self.create_subscription(
    Int32MultiArray, '/selected_tables', self.table_selection_callback, qos_profile
)

self.get_logger().info("Subscribed to /selected_tables")
```

reliability=QoSReliabilityPolicy.RELIABLE
→ 네트워크 품질이 낮더라도 반드시 메시지가 전달되도록 보장.

/selected_tables(조리 완료 테이블) 토픽을 데이터 손상을 최소화해 받도록 합니다

서빙 로봇_action

```
    def __init__(self):
        super().__init__('waypoint_follower')
        self.action_client = ActionClient(self, FollowWaypoints, '/follow_waypoints')

        # 1~9번 테이블 좌표 매핑 (x, y, z, w)
        self.table_positions = {
            1: (3.1855220365724546, 1.304425254256545, 0.1388262451047072, 0.9903167542105544),
            2: (3.1423153122211214, 0.20242532246961795, -0.13972915218417545, 0.9901897616264731),
            3: (3.1387965927306825, -0.8759655575738853, -0.08742478476855145, 0.9961711233559084),
            4: (1.9815345891885203, 1.1707028081502036, 0.20058184536500132, 0.9904124255009876),
            5: (2.081266442423764, 0.1124472149662954, 0.10934703761372845, 0.9898138412586049),
            6: (2.0263724508136813, -0.9178459014162126, -0.1669634024260172, 0.990694880273216),
            7: (0.8760495496863083, 1.3392586152477153, 0.15328578833779116, 0.9881818998006703),
            8: (0.9056097479873586, 0.1988398452508131 6, -0.9831499387987137, 0.1828009787722303),
            9: (0.9312649824286987, -0.9836673170471594, 0.13485513174136982, 0.4767080388769777),
        }

        # 출발점 좌표
        self.start_position = (0.0, 0.0, 0.0, 1.0)#회전이 없는 상태

        # 이동 상태 플래그
        self.is_moving = False
```

```
# 이동 시작
self.is_moving = True
goal_msg = FollowWaypoints.Goal()
goal_msg.poses = waypoints_list

self.action_client.wait_for_server()
self._send_goal_future = self.action_client.send_goal_async(goal_msg, feedback_callback=self.feedback_callback)
self._send_goal_future.add_done_callback(self.goal_response_callback)
```

`FollowWaypoints` 액션을 사용하여 여러 지점을 순차적으로 이동.

순서: 경로 준비(poseStamped) ->목표 전송->`send_goal_async()`를 통해 비동기적으로 액션을 요청.

self.get_logger().info(f'Current Waypoint Index: {feedback.current_waypoint}')로 현재 위치 확인

서빙 로봇_Service

```
# 🚀 서비스 추가
self.srv_get_position = self.create_service(Empty, 'get_robot_position', self.get_robot_position_callback)
self.srv_get_status = self.create_service(Empty, 'get_robot_status', self.get_robot_status_callback)
self.srv_cancel_movement = self.create_service(Empty, 'cancel_robot_movement', self.cancel_robot_movement_callback)
```

`FollowWaypoints` 액션을 사용하여 여러 지점을 순차적으로 이동.

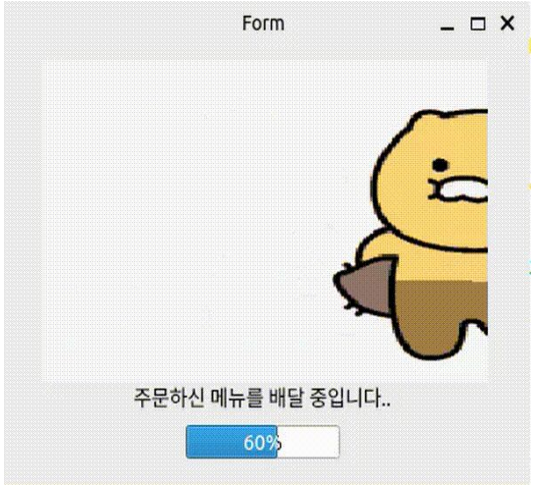
순서: 경로 준비(poseStamped) ->목표 전송->`send_goal_async()`를 통해 비동기적으로 액션을 요청.

self.get_logger().info("WaypointFollower services initialized!")를 통해 로봇의 위치,이동 상태,이동 취소

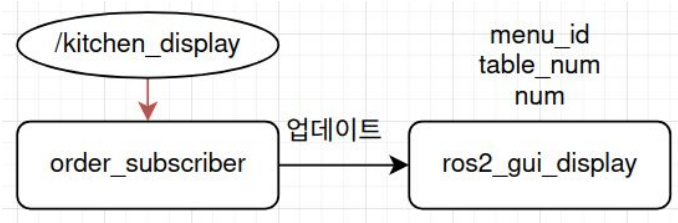
서빙 로봇 UI

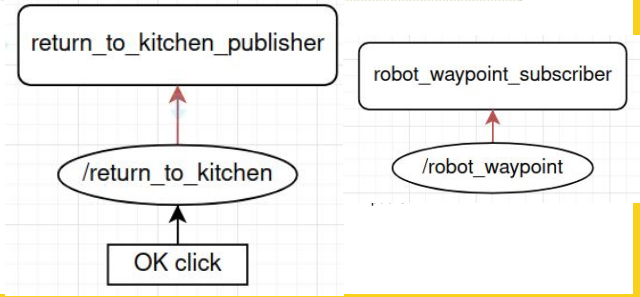
1. 대기 중

2. 배달 중



3.고객이 직접 로봇 복귀 명령





보완 사항

GUI

- 테이블 오더- 나의 주문건 진행상황 확인

  주문 접수, 조리 시작, 조리 완료 및 로봇 배차 대기,배달 시작
- 로봇 GUI 구현
- 다양한 UI 디자인 툴 사용

Database

감사합니다