Here is the complete, consolidated guide to building your **Grocery POS Kiosk** on the Jetson Orin Nano.

This guide combines the **optimized Python Backend** (using TensorRT + State Machine Logic) and the **responsive Frontend** (using flexbox layout and JSON-based overlay).

Project: Grocery AI Kiosk (Jetson Orin Nano)

1. System Preparation

First, ensure your Jetson is ready. Open a terminal and run the following to install dependencies globally (no virtual environment needed, as per your request).

```
# Update System
sudo apt update
sudo apt install python3-pip python3-dev libopencv-dev flatpak
matchbox-window-manager -y

# Install AI & Web Libraries
pip3 install fastapi uvicorn websockets mediapipe
```

Create your project structure:

```
mkdir -p ~/kiosk_project/backend
mkdir -p ~/kiosk_project/frontend
mkdir -p ~/kiosk_project/scripts
```

2. The Backend (pos_server.py)

This script manages the AI logic. It prioritizes the TensorRT engine (best.engine) for speed but falls back to .pt if needed. It implements the "Safety Lock" (disabling gestures when items are visible) and the "Grab" trigger (Open Hand \to Fist).

Create the file: nano ~/kiosk_project/backend/pos_server.py

Paste the code:

```
import cv2
import numpy as np
import mediapipe as mp
import json
import time
from fastapi import FastAPI, WebSocket, WebSocketDisconnect
from ultralytics import YOLO

app = FastAPI()

# --- 1. SETUP AI MODELS ---
print("Loading Grocery POS Models...")
```

```
# Load Object Detection (Prioritize TensorRT Engine)
try:
    model = YOLO('best.engine', task='detect')
    print("🚀 Custom TensorRT Engine Loaded (Fast Mode)")
except Exception as e:
    print(f"⚠️ Engine load failed ({e}). Trying .pt format...")
    try:
        model = YOLO('best.pt')
        print("✅ Custom Torch Model Loaded")
    except:
        print("⚠️ Custom model not found. Using Standard YOLO.")
        model = YOLO('yolov8n.pt')

# Load Hand Gesture Recognition
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(
    static_image_mode=False,
    max_num_hands=1,
    min_detection_confidence=0.5,
    min_tracking_confidence=0.5
)

# --- 2. PRODUCT DATABASE ---
# Ensure these IDs match your trained classes!
ITEM_DB = {
    0: {"name": "Nescafe Coffee", "price": 12.00},
    1: {"name": "Kopiko Coffee", "price": 15.00},
    2: {"name": "Lucky Me Pancit", "price": 25.00},
    3: {"name": "Coke in Can", "price": 45.00},
    4: {"name": "Alaska Milk", "price": 55.00},
    5: {"name": "Century Tuna", "price": 42.00},
    6: {"name": "VCut Spicy BBQ", "price": 38.00},
    7: {"name": "Selecta Cornetto", "price": 30.00},
    8: {"name": "Nestle Yogurt", "price": 35.00},
    9: {"name": "Femme Tissue", "price": 20.00},
    10: {"name": "Maya Champorado", "price": 40.00},
    11: {"name": "J&J Potato Chips", "price": 35.00},
    12: {"name": "Nivea Deodorant", "price": 89.00},
    13: {"name": "UFC Canned Mushroom", "price": 32.00},
    14: {"name": "Libby's Sausage", "price": 50.00},
    15: {"name": "Stik-O", "price": 65.00},
    16: {"name": "Nissin Cup Noodles", "price": 28.00},
    17: {"name": "Dewberry Strawberry", "price": 75.00},
    18: {"name": "Smart-C", "price": 35.00},
    19: {"name": "Pineapple Juice", "price": 40.00},
    20: {"name": "Nestle Chuckie", "price": 32.00},
    21: {"name": "Delight Probiotic", "price": 10.00},
    22: {"name": "Summit Water", "price": 20.00},
    23: {"name": "Almond Milk", "price": 120.00},
    24: {"name": "Piknik", "price": 85.00},
    25: {"name": "Bactidol", "price": 150.00},
    26: {"name": "Head & Shoulders", "price": 12.00},
    27: {"name": "Irish Spring Soap", "price": 45.00},
    28: {"name": "C2 Green Tea", "price": 28.00},
    29: {"name": "Colgate Toothpaste", "price": 95.00},
    30: {"name": "555 Sardines", "price": 22.00},
    31: {"name": "Meadows Truffle Chips", "price": 140.00},
    32: {"name": "Double Black", "price": 60.00},
    33: {"name": "Nongshim Noodles", "price": 55.00},
}

# --- 3. HELPER FUNCTIONS ---
def get_gesture_state(hand_landmarks):
    tips = [8, 12, 16, 20]
    pips = [6, 10, 14, 18]
    extended_count = 0
    for i in range(4):
        if hand_landmarks.landmark[tips[i]].y <
hand_landmarks.landmark[pips[i]].y:
            extended_count += 1

    if extended_count >= 4: return "OPEN"
    elif extended_count == 0: return "CLOSED"
    else: return "UNKNOWN"

# --- 4. STATE MANAGEMENT ---
class KioskState:
    def __init__(self):
        self.mode = "IDLE"
        self.cart = []
        self.total = 0.0
        self.last_scan_time = 0
        self.cooldown = 2.5
        self.last_gesture = "UNKNOWN"
        self.gesture_debounce = 0

state = KioskState()

# --- 5. WEBSOCKET LOOP ---
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    print("POS Client Connected")

    try:
        while True:
            # A. Receive Image
            data = await websocket.receive_bytes()
            np_arr = np.frombuffer(data, np.uint8)
            frame = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

            # B. Prepare Response
            response = {
                "mode": state.mode,
                "feedback": "",
                "cart": state.cart,
                "total": state.total,
                "boxes": [],
                "hand_coords": [],
                "gesture": "UNKNOWN"
            }

            # --- C. OBJECT DETECTION (Run First!) ---
            product_detected = False

            # Check for items in SCANNING mode (Safety Lock)
            if state.mode == "SCANNING":
                 results = model(frame, conf=0.6, verbose=False)
                 for r in results:
                    if len(r.boxes) > 0:
                        product_detected = True

                    for box in r.boxes:
                        x1, y1, x2, y2 = box.xyxy[0].tolist()
                        label_name = model.names[int(box.cls[0])]

                        response["boxes"].append({
                            "coords": [x1, y1, x2, y2],
                            "label": label_name
                        })

                        # Add to Cart Logic
                        cls_id = int(box.cls[0])
                        if cls_id in ITEM_DB:
                            item = ITEM_DB[cls_id]
                            scan_time = time.time()
                            if scan_time - state.last_scan_time >
state.cooldown:
                                state.cart.append(item)
                                state.total += item["price"]
                                state.last_scan_time = scan_time
                                response["feedback"] = f"Added
{item['name']}!"
```

```
            # --- D. HAND GESTURE DETECTION ---
            current_gesture = "UNKNOWN"

            # CRITICAL CHECK: Only run gesture logic if NO product is
detected
            if not product_detected:
                frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                hand_results = hands.process(frame_rgb)

                if hand_results.multi_hand_landmarks:
                    hand_lms = hand_results.multi_hand_landmarks[0]
                    # Save coords for drawing
                    for lm in hand_lms.landmark:
                        response["hand_coords"].append([lm.x, lm.y])

                    current_gesture = get_gesture_state(hand_lms)
                    response["gesture"] = current_gesture

            # --- E. GESTURE TRANSITION LOGIC ---
            trigger_action = False
            curr_time = time.time()

            if current_gesture != "UNKNOWN":
                if curr_time - state.gesture_debounce > 1.5:
                    if state.last_gesture == "OPEN" and
current_gesture == "CLOSED":
                        trigger_action = True
                        state.gesture_debounce = curr_time
                        print("👉 GRAB DETECTED!")
                state.last_gesture = current_gesture
            else:
                state.last_gesture = "UNKNOWN"

            # --- F. STATE MACHINE ---
            if state.mode == "IDLE":
                response["feedback"] = "Clench Fist to Start ✊"
                if trigger_action:
                    state.mode = "SCANNING"
                    state.cart = []
                    state.total = 0.0
                    print("Transition: IDLE -> SCANNING")

            elif state.mode == "SCANNING":
                if not product_detected:
                     if response["feedback"] == "":
response["feedback"] = "Scanning... Clench to Pay ✊"
```

```
                if trigger_action:
                    state.mode = "PAID"
                    print("Transition: SCANNING -> PAID")

            elif state.mode == "PAID":
                response["feedback"] = "Paid! Resetting..."
                if time.time() - state.gesture_debounce > 4.0:
                     state.mode = "IDLE"

            # G. Send JSON
            await websocket.send_text(json.dumps(response))

    except WebSocketDisconnect:
        print("Client Disconnected")
        state.mode = "IDLE"
    except Exception as e:
        print(f"Error: {e}")
```

3. The Frontend (pos_kiosk.html)

This uses the responsive design logic. It fills the available screen space, uses the Peso currency sign, and displays the "safety lock" feedback implicitly by not showing gesture skeletons when items are present.
Create the file: nano ~/kiosk_project/frontend/pos_kiosk.html
Paste the code:

```
<!DOCTYPE html>
<html>
<head>
    <title>Jetson Grocery POS</title>
    <style>
        body { margin: 0; background: #222; font-family: 'Segoe UI',
sans-serif; color: white; display: flex; height: 100vh; overflow:
hidden; }

        /* LAYOUT */
        #main-view {
            flex: 1;
            position: relative;
            background: #000;
            display: flex;
            justify-content: center;
            align-items: center;
        }

        #sidebar {
            width: 400px;
            min-width: 350px;
            background: #333;
            padding: 20px;
            display: flex;
            flex-direction: column;
            box-shadow: -5px 0 15px rgba(0,0,0,0.5);
            z-index: 2;
        }

        /* VIDEO CONTAINER */
        #container { position: relative; width: 100%; height: 100%; }
        video { width: 100%; height: 100%; object-fit: contain; }
        #overlay { position: absolute; top: 0; left: 0; width: 100%;
height: 100%; pointer-events: none; }

        /* UI ELEMENTS */
        h1 { margin: 0 0 20px 0; color: #4CAF50; border-bottom: 2px
solid #555; padding-bottom: 10px; }

        #item-list { flex-grow: 1; overflow-y: auto; list-style: none;
padding: 0; margin: 0; }
        .item-row { display: flex; justify-content: space-between;
padding: 12px 5px; border-bottom: 1px solid #444; font-size: 1.2rem; }
        .price { font-weight: bold; color: #ddd; }

        #total-box { font-size: 2.5rem; font-weight: bold; text-align:
right; margin-top: auto; padding-top: 20px; border-top: 3px solid
#fff; color: #4CAF50; }

        #msg-box {
            position: absolute; bottom: 30px; left: 50%; transform:
translateX(-50%);
            background: rgba(0,0,0,0.8); padding: 15px 40px;
border-radius: 50px;
            font-size: 1.5rem; text-align: center; white-space:
nowrap; transition: color 0.3s;
            z-index: 10;
        }

        /* Hidden Canvas for capture */
        #capture { display: none; }
    </style>
</head>
<body>

<div id="main-view">
    <div id="container">
        <video id="cam" autoplay playsinline muted></video>
        <canvas id="overlay"></canvas>
    </div>
    <div id="msg-box">Connecting...</div>
</div>

<div id="sidebar">
    <h1>🛒 My Cart</h1>
    <ul id="item-list"></ul>
    <div id="total-box">Total: ₱0.00</div>
</div>

<canvas id="capture"></canvas>

<script>
    const video = document.getElementById('cam');
    const overlay = document.getElementById('overlay');
    const capture = document.getElementById('capture');
    const msgBox = document.getElementById('msg-box');
    const itemList = document.getElementById('item-list');
    const totalBox = document.getElementById('total-box');

    const ctxOverlay = overlay.getContext('2d');
    const ctxCapture = capture.getContext('2d');

    let socket = null;
    let isProcessing = false;
    let resetTimer = null;

    // Internal resolution for AI (Faster transmission)
    const AI_WIDTH = 640;
    const AI_HEIGHT = 480;

    // 1. Start Camera
    async function start() {
        try {
            const stream = await navigator.mediaDevices.getUserMedia({
                video: { width: { ideal: 1280 }, height: { ideal: 720
} }, audio: false
            });
            video.srcObject = stream;
            video.onloadedmetadata = () => {
                video.play();
                capture.width = AI_WIDTH;
                capture.height = AI_HEIGHT;
                resizeOverlay();
                window.addEventListener('resize', resizeOverlay);
                connect();
            };
        } catch(e) {
            msgBox.innerText = "Camera Error";
            msgBox.style.color = "red";
        }
    }

    function resizeOverlay() {
        overlay.width = video.clientWidth;
        overlay.height = video.clientHeight;
    }

    // 2. Connect
    function connect() {
        socket = new WebSocket("ws://127.0.0.1:8080/ws");

        socket.onopen = () => {
            msgBox.innerText = "Connected!";
            requestAnimationFrame(sendFrame);
        };

        socket.onmessage = (event) => {
            const data = JSON.parse(event.data);

            // A. Update Status Text
            msgBox.innerText = data.feedback || data.mode;
            if (data.mode === "SCANNING") msgBox.style.color =
"#4CAF50";
            else if (data.mode === "PAID") msgBox.style.color =
"cyan";
            else msgBox.style.color = "white";

            // B. Draw Overlays
            ctxOverlay.clearRect(0, 0, overlay.width, overlay.height);
            drawOverlays(data);

            // C. Update Receipt
            updateReceipt(data.cart, data.total);

            // D. Handle PAID Reset Logic
            if (data.mode === "PAID" && !resetTimer) {
                resetTimer = setTimeout(() => { resetTimer = null; },
4000);
            }

            isProcessing = false;
            requestAnimationFrame(sendFrame);
        };
        socket.onclose = () => setTimeout(connect, 3000);
    }

    function sendFrame() {
        if (!socket || socket.readyState !== WebSocket.OPEN ||
isProcessing) return;
        isProcessing = true;
        ctxCapture.drawImage(video, 0, 0, AI_WIDTH, AI_HEIGHT);
        capture.toBlob(blob => socket.send(blob), 'image/jpeg', 0.5);
    }

    function drawOverlays(data) {
        const scaleX = overlay.width / AI_WIDTH;
        const scaleY = overlay.height / AI_HEIGHT;

        // 1. Hands
        if (data.hand_coords && data.hand_coords.length > 0) {
            ctxOverlay.fillStyle = data.gesture === "OPEN" ? "#0f0" :
"red";
            data.hand_coords.forEach(pt => {
                ctxOverlay.beginPath();
                ctxOverlay.arc(pt[0] * overlay.width, pt[1] *
overlay.height, 5, 0, 2*Math.PI);
                ctxOverlay.fill();
            });
        }

        // 2. Boxes
        ctxOverlay.lineWidth = 3;
        ctxOverlay.font = "bold 18px Arial";

        data.boxes.forEach(box => {
            const [rawX1, rawY1, rawX2, rawY2] = box.coords;

            const x1 = rawX1 * scaleX;
            const y1 = rawY1 * scaleY;
            const x2 = rawX2 * scaleX;
            const y2 = rawY2 * scaleY;

            ctxOverlay.strokeStyle = "#00FFFF";
            ctxOverlay.strokeRect(x1, y1, x2 - x1, y2 - y1);

            ctxOverlay.fillStyle = "#00FFFF";
            const text = box.label;
            const width = ctxOverlay.measureText(text).width;
            ctxOverlay.fillRect(x1, y1 - 25, width + 10, 25);
            ctxOverlay.fillStyle = "black";
            ctxOverlay.fillText(text, x1 + 5, y1 - 6);
        });
    }

    function updateReceipt(cart, total) {
        itemList.innerHTML = "";
        cart.forEach(item => {
            const li = document.createElement("li");
            li.className = "item-row";
            li.innerHTML = `<span>${item.name}</span> <span 
class="price">₱${item.price.toFixed(2)}</span>`;
            itemList.appendChild(li);
        });
        itemList.scrollTop = itemList.scrollHeight;
        totalBox.innerText = "Total: ₱" + total.toFixed(2);
    }

    start();
</script>
</body>
</html>
```

4. The Model File

Make sure your TensorRT model is in the right place.

1. **If you already exported it:** Move your best.engine file to ~/kiosk_project/backend/.
2. **If you haven't exported it:** Run this command in ~/kiosk_project/backend/ (assuming you have best.pt there):
   ```
   yolo export model=best.pt format=engine device=0 half=True
   ```

5. Startup Script (start_kiosk.sh)

This script automates the launch process. It starts the Window Manager, the Backend, waits for models to load, and then launches the Frontend.

Create the file: nano ~/kiosk_project/scripts/start_kiosk.sh

Paste the code:

```
#!/bin/bash
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" &>/dev/null && pwd)"
PROJECT_ROOT="${SCRIPT_DIR}/.."

# 1. Maximize Performance (if needed)
# sudo jetson_clocks

# 2. Start Window Manager (prevents UI glitches)
matchbox-window-manager &
```

```
# 3. Start Python Backend (Background)
echo "Starting Backend..."
python3 "${PROJECT_ROOT}/backend/pos_server.py" >
"${PROJECT_ROOT}/backend.log" 2>&1 &
BACKEND_PID=$!

# Wait for models to load (Give it 5s for TensorRT engine)
sleep 5

# 4. Start Chromium in Kiosk Mode
echo "Starting Kiosk..."
flatpak run org.chromium.Chromium \
  --kiosk \
  --incognito \
  --disable-infobars \
  --noerrdialogs \
  --disable-pinch \
  --use-fake-ui-for-media-stream \
  --check-for-update-interval=31536000 \
  "file://${PROJECT_ROOT}/frontend/pos_kiosk.html"

# 5. Cleanup
kill $BACKEND_PID
```

Make it executable:

```
chmod +x ~/kiosk_project/scripts/start_kiosk.sh
```

6. How to Run

Manual Test:

```
~/kiosk_project/scripts/start_kiosk.sh
```

User Workflow:

1. **Idle Mode:** Screen says **"Clench Fist to Start ✊"**.
   - *Action:* Show open hand ✋ \to Close fist ✊.
2. **Scanning Mode:** Screen says **"Scanning..."**.
   - *Action:* Hold up grocery items. You will see bounding boxes and they will appear on the receipt.
   - *Note:* If you hold up an item, hand gestures are disabled (Safety Lock).
3. **Payment Mode:** Screen says **"Clench to Pay ✊"**.
   - *Action:* Put down items. Show open hand ✋ \to Close fist ✊.
   - *Result:* "Paid! Resetting..." message appears, and the kiosk resets after 4 seconds.