BRAUDE
College of Engineering, karmiel

Capstone Project – Phase A

# Personalized Smart Cart with AI Assistance

25-2-D-23

Tomer Rotman

David Zorin

**Supervisor**
Uzi Rosen

# Table of Contents


**3 Problem Definition**

**5 Crafting the Solution**

6 Online Shopping

7 Smart Cart Systems

8 Market Analysis

9 Proposed Solution

11 Enabling Technologies

13 Hardware Components

14 Software Components

**15 Engineering Process**

16 Technical Challenges

17 Computing Unit and OS

18 Voice Assistant

19 Alternatives Suggestions

20 Layout Visualization

21 Route Optimization

22 OR-Tools Evaluation

23 Development Process

**24 Work Artifacts**

25 Primary Requirements

27 Use Case Diagram

28 Activity Diagram

29 Software Architecture

30 User Interface

**36 Verification & Evaluation**

37 Evaluation

38 Verification

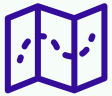
# 01

# Problem Definition

# Problem Definition

Grocery shopping is **time-consuming**, why?

Struggle to Read Nutritional Information



Inefficient Supermarket Traversal

Tendency to Forget

Long Lines at Checkout Counters

# 02

# Crafting the Solution

# Online Shopping

**84% of all retail sales still occur in physical stores**

Innovation must be brought into them

# Smart Cart Systems

Remove the need to scan items at checkout counters

Long Lines at
Checkout Counters


Typically, IoT-based system:

- Touchscreen
- Barcode scanners and/or cameras
- Weighing sensors
- Card terminal

# Market Analysis

A comparison on three smart cart systems was conducted.

| Smart Cart | Key Features | Observation |
|---|---|---|
| Shopic | Call staff for assistance | Staff often doesn't respond |
| Cust²Mate | Product location via name typing | Not ideal for those who struggle with tech |
| | Shopping list Synchronization | Items are still picked up in a scattered order |
| SUPERSMART | Retail analytics tools | Nothing particularly useful for consumers |

**All smart carts provide recommendations.**

# Proposed Solution

A **smart cart integrated with a digital companion.** capabilities:

**Natural Language Interactions**

**Nutritional Guidance**

Answering questions about the nutritional content, allergen presence and ethical alignment of a scanned product

**Alternatives Suggestions**

Offering alternatives based on mentioned criteria

**Product Location Guidance**

Verifying product availability and displaying his location on a map

# Proposed Solution (Continued)

More capabilities:

### Warnings on Conflicts

Flagging scanned products that conflicts with stated concerns while automatically providing alternatives

### Route Optimization

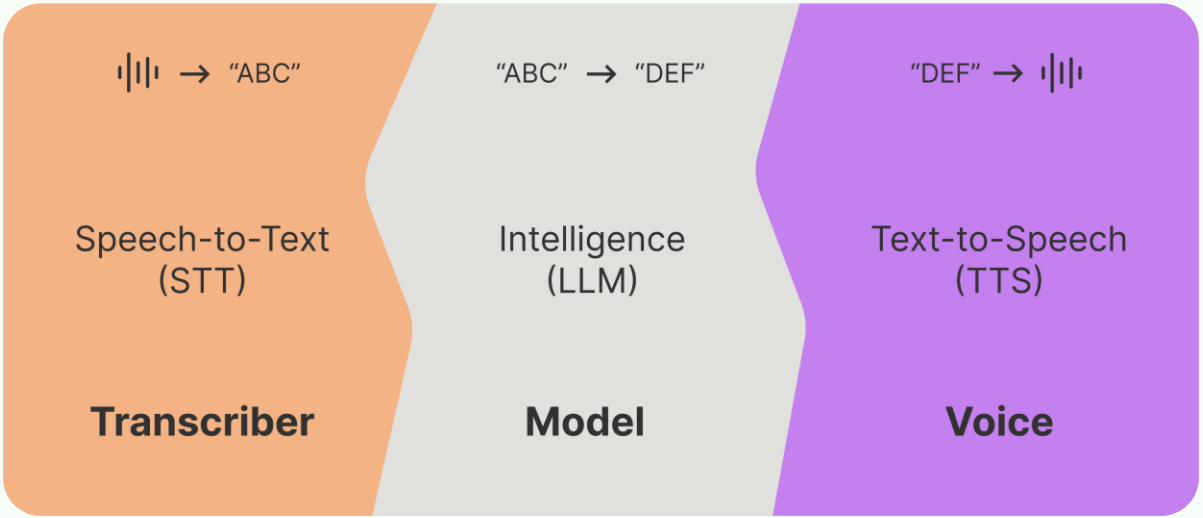
Displaying a near-optimal pickup order on a map according to pre-created shopping list

### Frequently-Bought Suggestions

Offering items might forgotten upon checkout

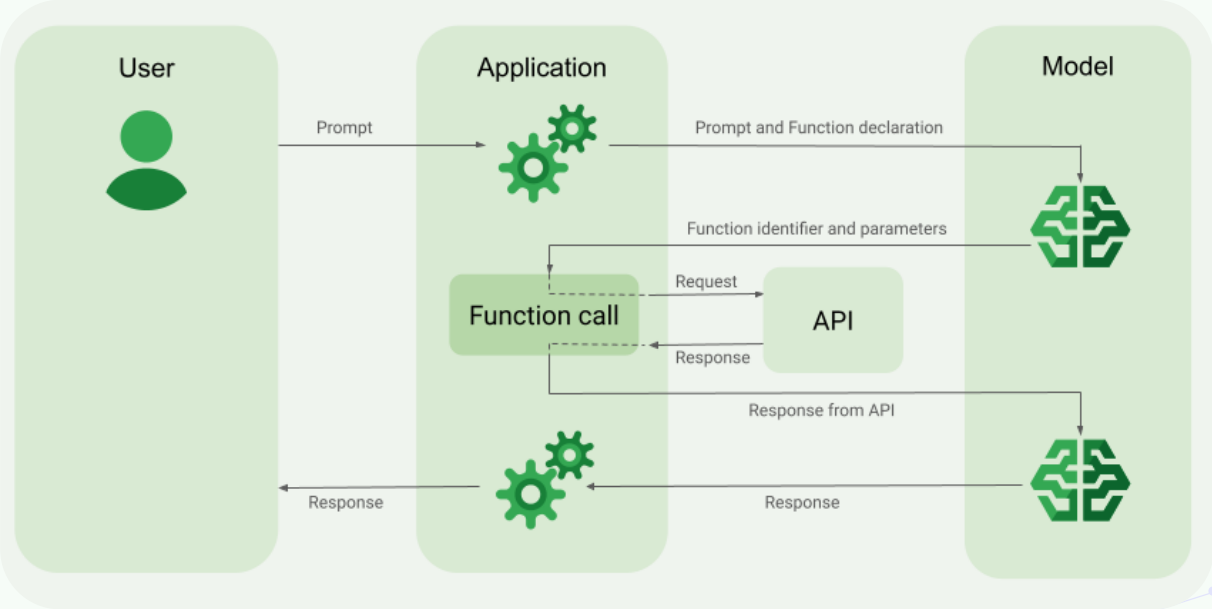
# Voice Assistant Enabling Technologies

Intelligent voice-based interactions became possible with the advancements of AI models



# Voice Assistant Enabling Technologies

Connecting the LLM to a database is enabled through the **Function Calling Method**



# Hardware Components

Based on the system's capabilities, these are the hardware components required to support them:

Computing Unit

Touchscreen Display

Barcode Scanner

Microphone

Speaker

Battery

# Software Components

And these are the necessary software components:

User Interface

Operating System

Voice-based AI Agent

Backend Modules

Database

Authentication Service

# 03

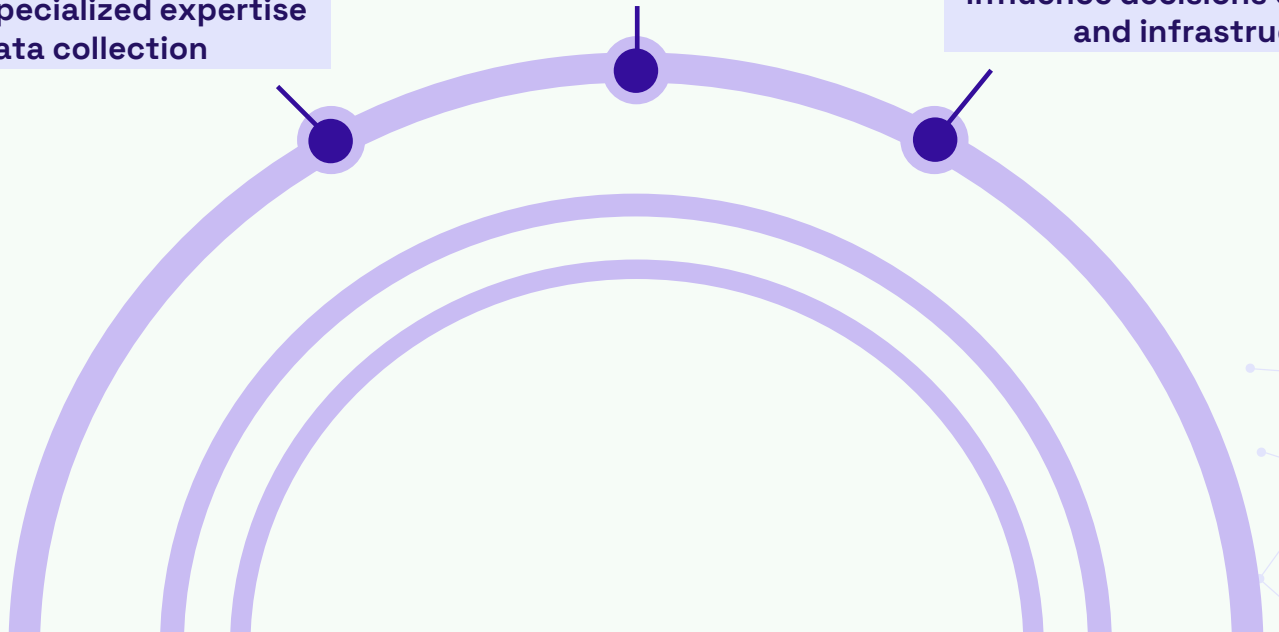
# Engineering Process

# Constraints and Technical Challenges

Fine-tuning an LLM requires substantial computational resources, specialized expertise and data collection

Implementing a production-level smart cart system carried by two students is not feasible

The prototype is subject to budgetary limitations that influence decisions on hardware and infrastructure



# Computing Unit and OS

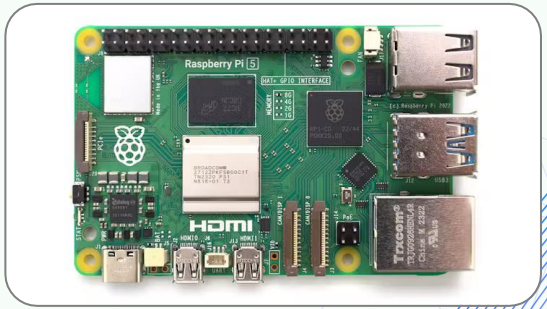
As SBCs are more suitable for the proposed system, we rely on **Raspberry Pi 5**.

Limitations:

- Lacks AI Acceleration hardware
- High power consumption

Installing **Raspberry Pi OS**:

- Better hardware access than **Android**
- Officially supported



# Voice Assistant

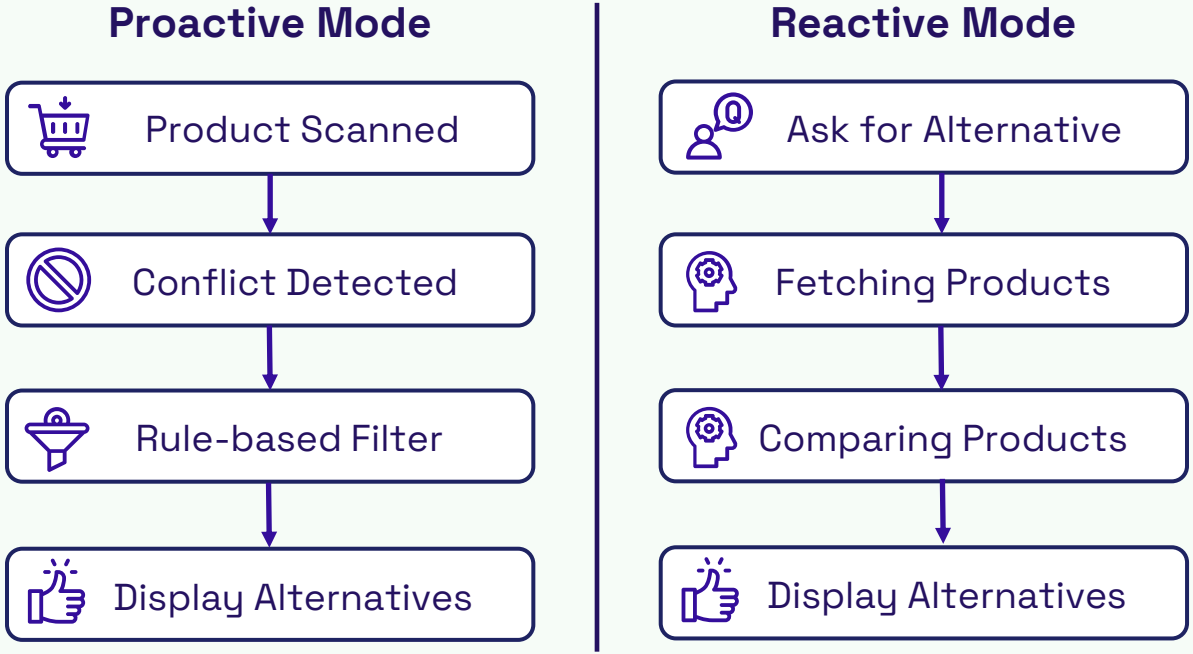
STT, LLM and TTS integration – engineering effort to optimize

- Latency
- Scalability (as the number of users grow)

Using **VAPI** – platform to build and deploy voice AI agents:

- Flexibility in choosing models
- Built-in function calling endpoints definition
- Ability to guide LLM behavior with prompt engineering

# Alternatives Suggestions



# Supermarket Layout Visualization

Using a matrix-based tile system:

- Each cell = path or obstacle tile
- Tile styled with **CSS**
- Location icons styled with **SVG**, placed on top
- Responsive

```
[[·,  ·,  #],
 [·,  ·,  #],
 [·,  #,  #]]
```

| | | |
|---|---|---|
| # | | Obstacle Tile |
| · | | Path Tile |

# Route Optimization

**Problem:** Find shortest route to reach all items once

**Equivalent to Traveling Salesman Problem:**
“Visit each city once, return to origin, minimize total distance”

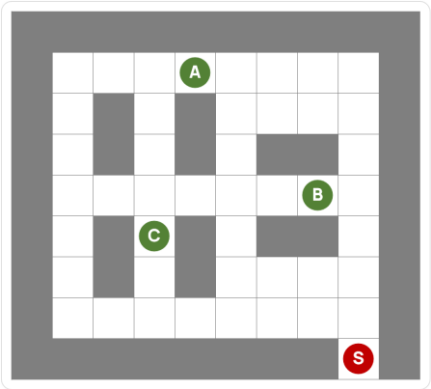
Utilizing **OR-Tools** by Google:

- Python Library
- Built in TSP-Solvers
- Near-optimal results

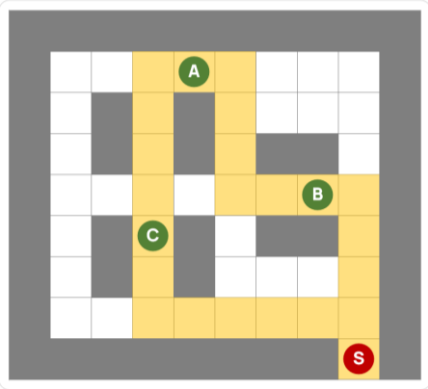
Input needed:

- Distance matrix between products on the list
- A distance matrix of all product distances on the supermarket is created once, filtered with **NumPy** each time list is created

# OR-Tools Evaluation




Simplified Supermarket Layout
‘A’, ‘B’, ‘C’ Represent Products




Optimal Solution

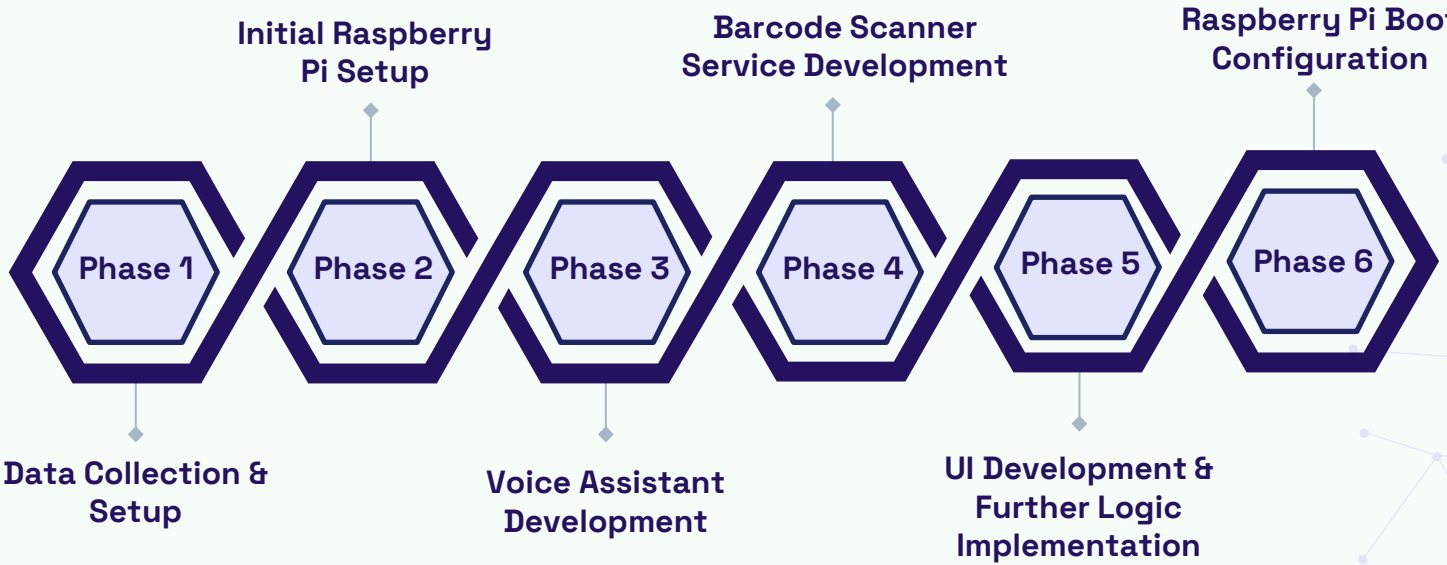
```python
# Problem data initialization
data = {}
data["distance_matrix"] = [
      [0, 12, 6, 9], # 'S' distances
      [12, 0, 7, 6], # 'A' distances
      [6, 7, 0, 6],  # 'B' distances
      [9, 6, 6, 0]   # 'C' distances
]
data["num_vehicles"] = 1 # Represents a single customer (TSP)
data["depot"] = 0        # Index of the start/end node ('S')

# Routing model creation
manager = pywrapcp.RoutingIndexManager(
    len(data["distance_matrix"]), data["num_vehicles"], data["depot"]
)
routing = pywrapcp.RoutingModel(manager)

# Callback: returns distance between two nodes
def distance_callback(from_index, to_index):
    # Convert from routing variable Index to distance matrix NodeIndex
    from_node = manager.IndexToNode(from_index)
    to_node = manager.IndexToNode(to_index)
    return data["distance_matrix"][from_node][to_node]

# Register the distance callback and set cost evaluator
transit_callback_index = routing.RegisterTransitCallback(distance_callback)
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

# Set search strategy: greedy initialization using PATH_CHEAPEST_ARC
search_parameters = pywrapcp.DefaultRoutingSearchParameters()
search_parameters.first_solution_strategy = (
    routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
)

# Solve the problem
solution = routing.SolveWithParameters(search_parameters)
if solution:
    total_distance = solution.ObjectiveValue()
    index = routing.Start(0)
    optimal_path = []
    while not routing.IsEnd(index):
        optimal_path.append(manager.IndexToNode(index))
        index = solution.Value(routing.NextVar(index))
    print(f"Near-optimal shopping path: {optimal_path}")
```

```
Near-optimal shopping path: [0, 2, 1, 3]
‘S’, ‘B’, ‘A’, ‘C’
```

# Development Process

Agile methodology is essential in this project.

While promotes iterative development, it's still helpful to outline key phases to guide the process:



# 04

# Work Artifacts

OTP – One Time Password

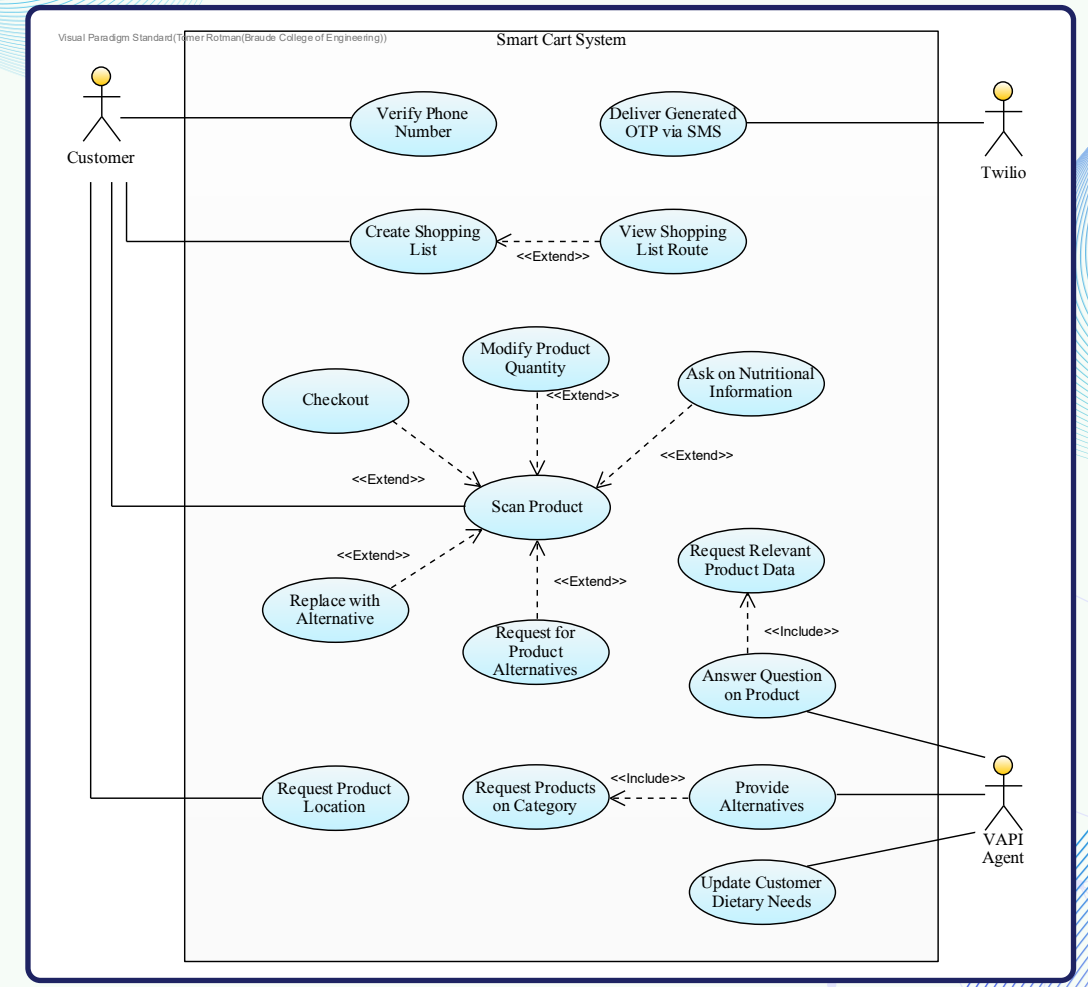
# Primary Requirements

| No. | FR / NFR | Requirement | Type |
|---|---|---|---|
| 1 | FR | The system shall support OTP verification for user authentication | |
| 1.1 | NFR | OTP is delivered via SMS | Security |
| 2 | FR | The system shall allow users to create a shopping list | |
| 2.1 | NFR | List creation is performed on external device | Interoperability |
| 3 | FR | The system shall display layout with near-optimized pickup order | |
| 3.1 | NFR | The pickup order is displayed when the user loads a pre-created list | Usability |
| 4 | FR | The system shall support AI-driven interactions for user inquiries | |
| 4.1 | NFR | Follow-up questions may be asked by the voice assistant to:<br>gather missing details, better understand user dietary needs | Adaptability |
| 5 | FR | The system shall provide personalized warnings along with suitable alt’ | |
| 6 | FR | The system shall suggest products that may have been forgotten | |

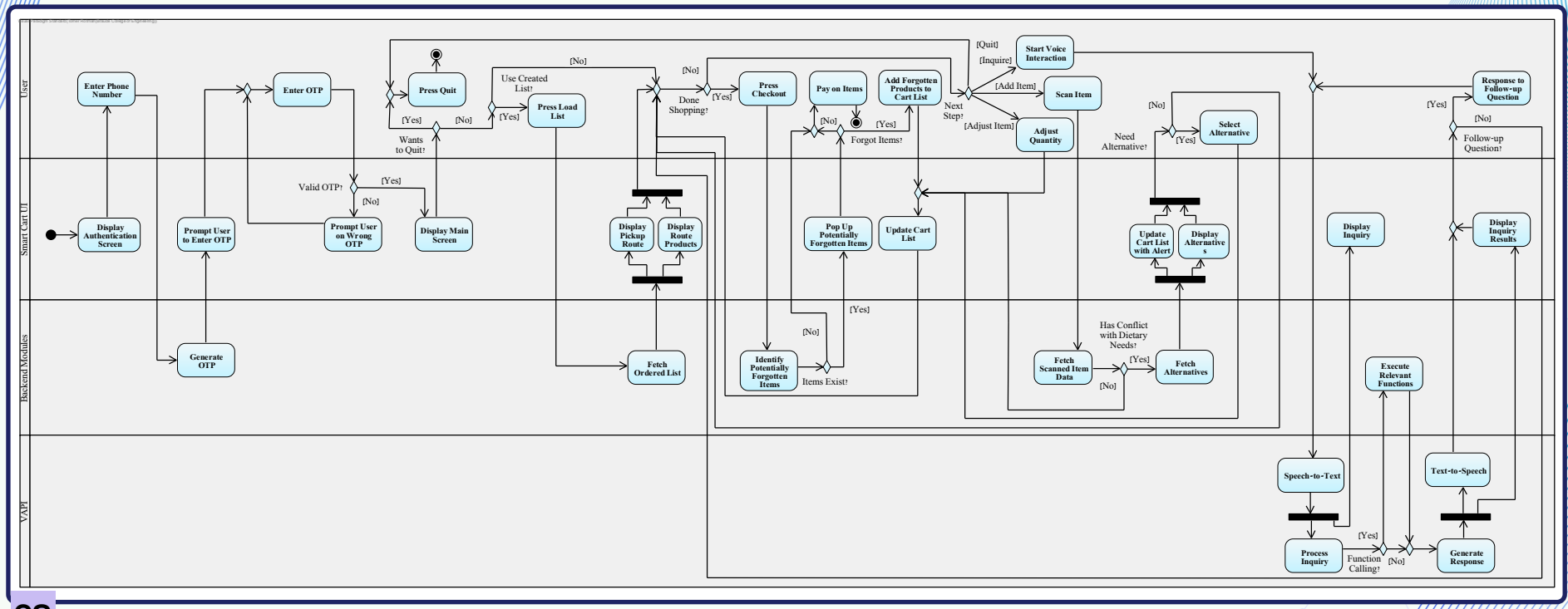
# Primary Requirements

| No. | FR / NFR | Requirement | Type |
|---|---|---|---|
| 7 | NFR | The interface shall automatically recover from unexpected crash with all loaded items | Fault Tolerance |
| 8 | NFR | The system shall be centrally updated | Maintainability |
| 9 | NFR | Concurrent usage by multiple smart carts shall not cause system shutdown | Stability |
| 10 | NFR | System responses shall be provided within reasonable time (< 3 seconds) | Performance |

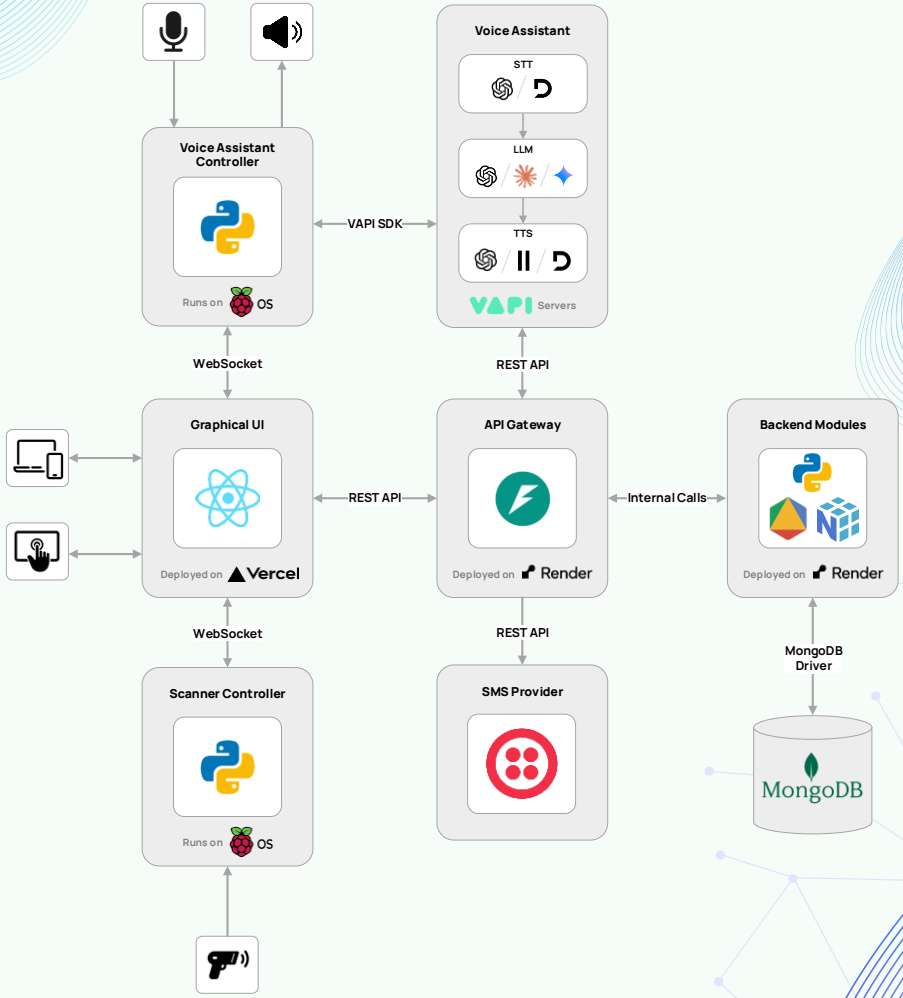
# Use Case Diagram



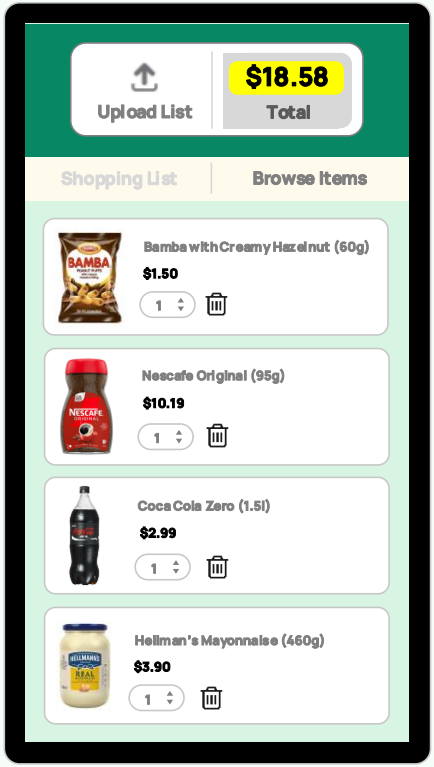
# Activity Diagram



# Software Architecture

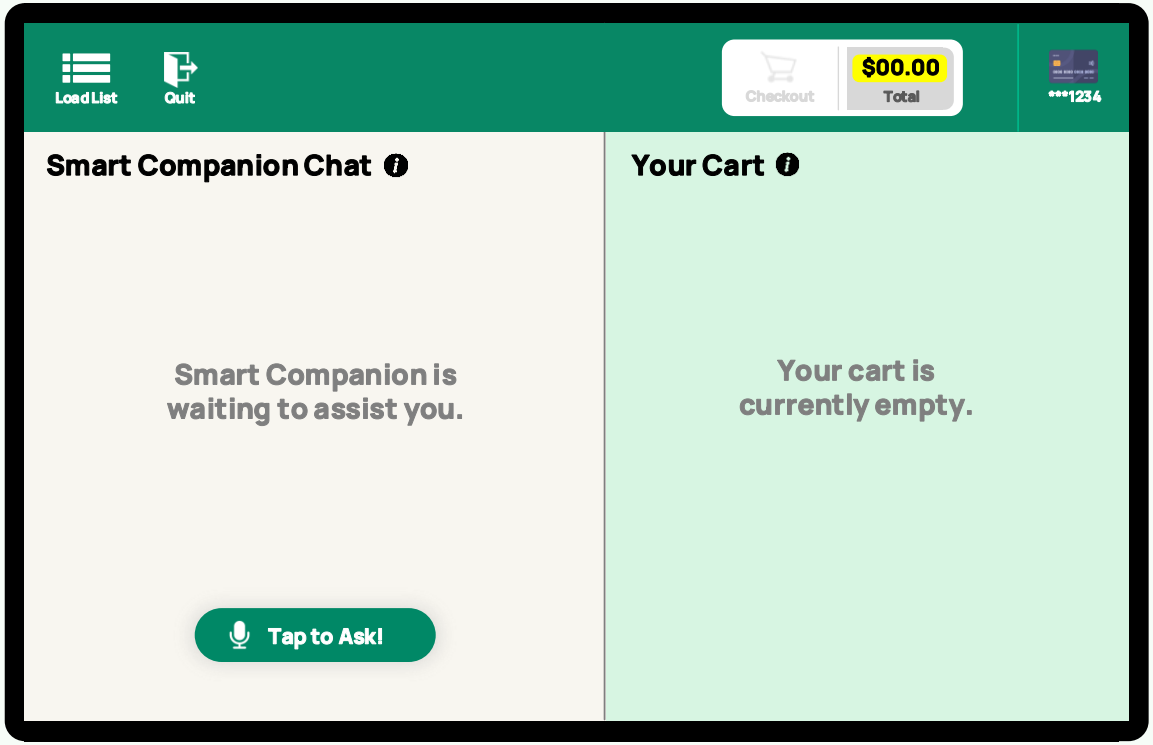


# User Interface



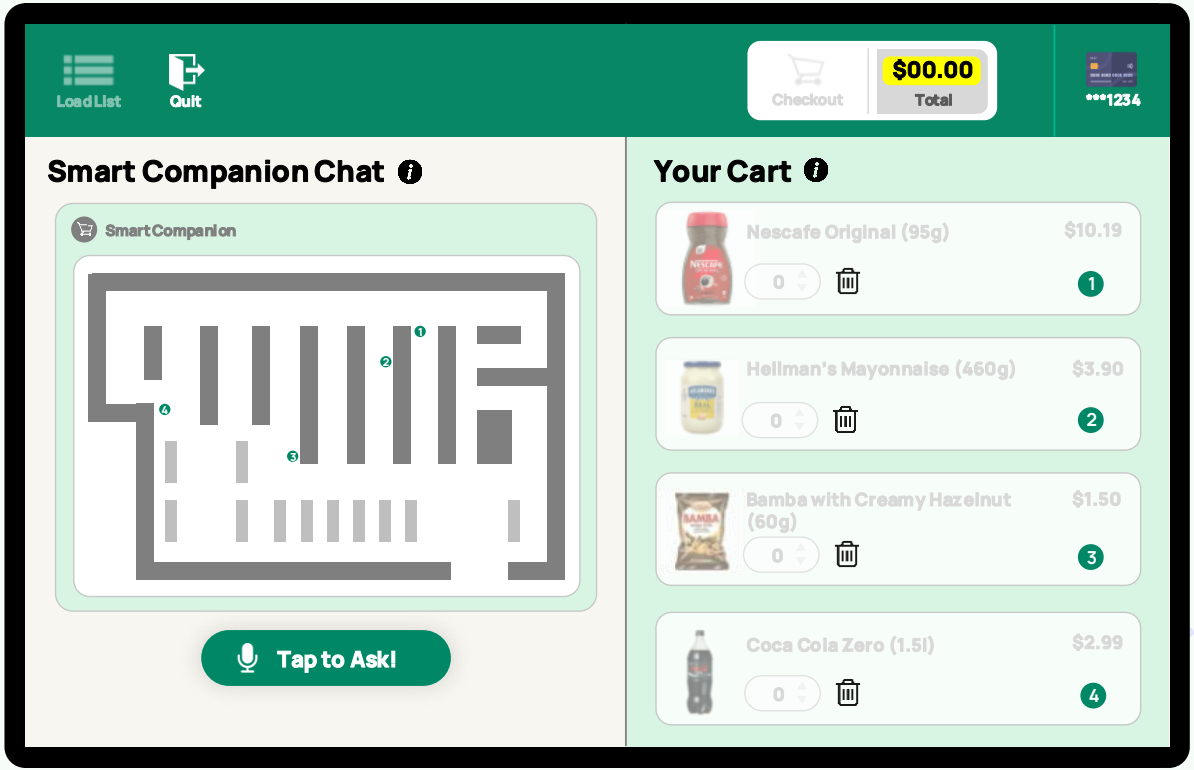

List Creation Interface

# User Interface



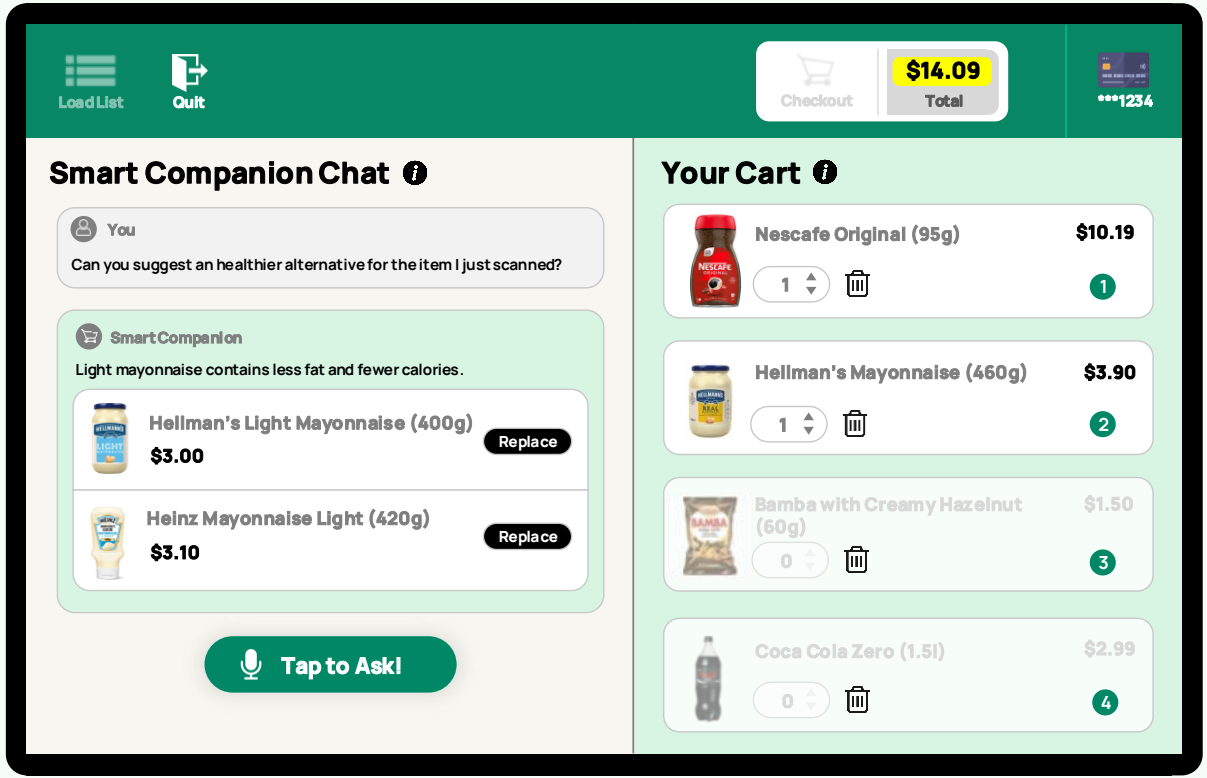

Smart Cart Interface

# User Interface



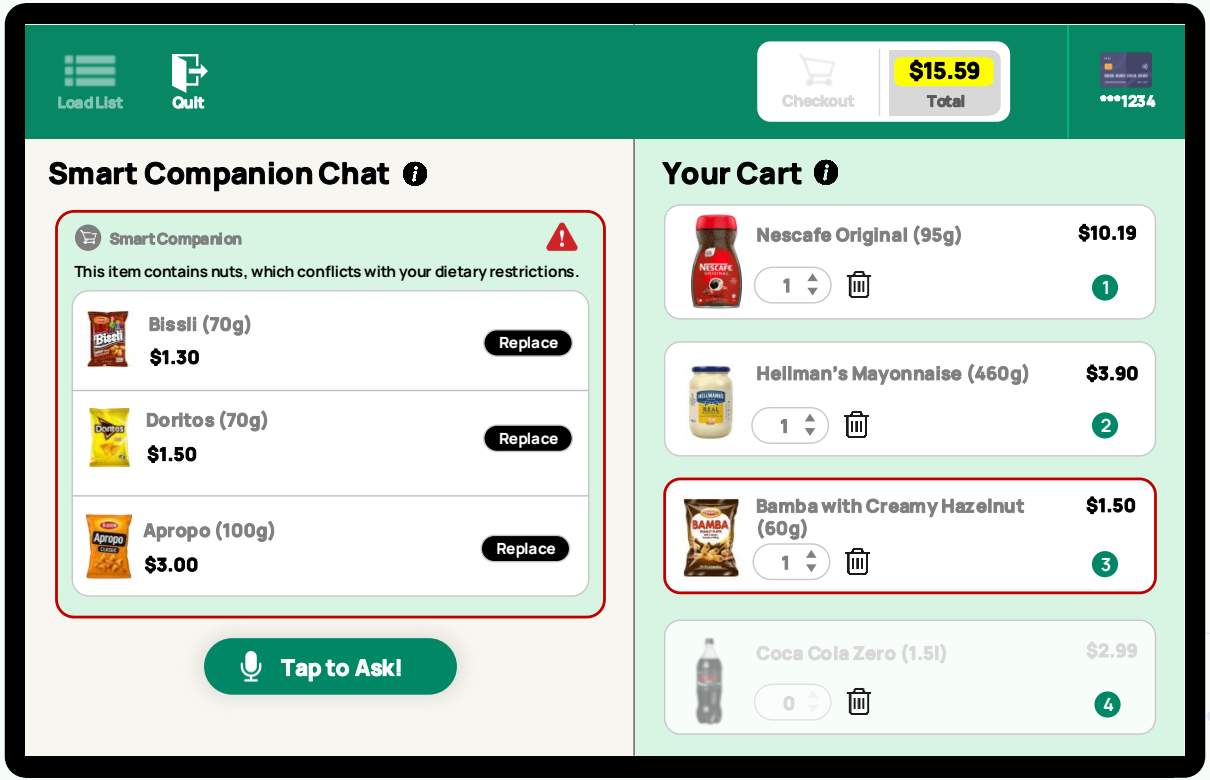

Load List

# User Interface



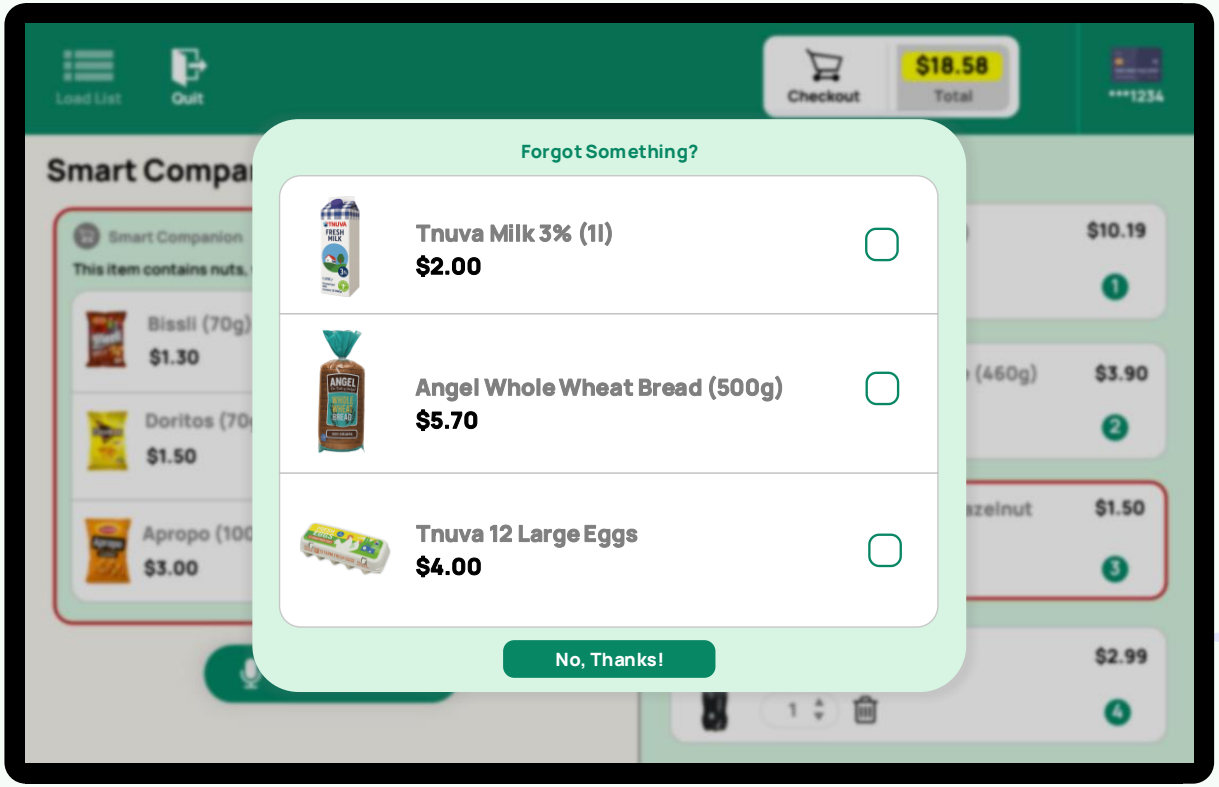

Alternative Suggestion Reactive Mode

# User Interface




Personalized Alerts

# User Interface




Frequently-Bought Suggestions

# 05

# Evaluation & Verification

# Success Criteria and Evaluation

| Criterion | Explanation | Evaluation |
|---|---|---|
| Cost Feasibility | Achieving AI functionality while maintaining realistic operational costs | Using VAPI analytics tools |
| Performance | AI Response Initiation < 3 seconds | |
| Usability | SUS > Average Score (68) | Participants will complete SUS questionnaire, and conversation will be documented to asses accuracy |
| Accuracy | Achieve intended response in at least 90% of user inquires | |
| Stability | Handle 3 parallel inquiries without crashing | Running multiple application instances in parallel |

SUS – System Usability Scale

# Verification

Summary of the key verification areas:

### Voice Assistant

Ability to handle different questions, ask for clarification when needed, politely reject out-of-scope inquiries.

### UI

Ensuring scanned products update cart instantly, quantities and costs adjusts accurately, UI elements are responsive

### List Creation

Large product lists (>50) generate near-optimized pickup order quickly

### User Authentication

Verify correct handling of new and returning users

### Boot and Recovery

Simulating Raspberry Pi crashes to confirm services auto-restart

# Thank you!