Software Engineering Department

Capstone Project - Phase A

Personalized Smart Cart with AI Assistance

25-2-D-23

Authors

Tomer Rotman
tomer.rotman@e.braude.ac.il

David Zorin
david.zorin@e.braude.ac.il

Supervisor

Uzi Rosen
rosen@braude.ac.il

GitHub Repository

Link

Table of Contents

Abstract

This project presents a Raspberry Pi-based smart cart system with a React frontend and FastAPI backend, integrated with an AI-driven voice assistant, powered by VAPI. The assistant can respond to inquiries about nutritional information, suggest alternative products based on specified needs for a scanned item, and intuitively display product locations on a visual map. It engages in follow-up questions to identify dietary restrictions and, from that point onward, automatically issues warnings about potential conflicts for all scanned products across shopping sessions, while also suggesting suitable alternatives - reducing the need for manual searches. Users can create shopping lists in advance using personal devices and upload them to the system, which generates an optimized pick-up route using OR-Tools and NumPy. Lastly, to reduce the risk of forgetting items, the system provides suggestions on frequently bought products during checkout. Altogether, these features enable a more personalized, informed and time-efficient shopping experience.

1 Introduction

Despite the rapid growth of online shopping, physical grocery stores continue to be the primary choice for many consumers around the world. In fact, 84% of all retail sales still occur in physical stores, partly due to the limitations of online shopping such as lack of immediacy (since customers must wait for delivery), uncertainty (as products cannot be seen or touched before purchase) and perceived risks (such as receiving items that do not match expectations) [1]. One of the most persistent challenges in traditional grocery shopping, however, is the problem of lengthy checkout lines, particularly during peak shopping periods such as weekends and holidays. Negative emotional responses to long waiting times have been shown to worsen the perceived store image [2], which refers to the overall perception that consumers form about a retail store. Since store image significantly influences customer loyalty [3], retailers are encouraged to seek innovative solutions in order to retain customer loyalty and expand their customer base.

Over the past two decades, the expansion of the Internet of Things (IoT) has driven a major technological revolution across multiple industries. In the retail sector, smart carts represent a specific application of IoT. These systems typically include a touchscreen, barcode scanners, and network connectivity, enabling them, through the touchscreen interface, to display the names and prices of the items selected by the consumer, along with the total cost of the purchase. When equipped with a card terminal, smart carts allow customers to complete payments directly at the cart – eliminating the need to wait in checkout lines. Even in the absence of integrated payment functionality, smart carts have the potential to eliminate queue times by removing the need to scan items at checkout. In such cases, checkout lines serve primarily as a verification stage to prevent theft, typically supported by weight sensors embedded along the checkout path to detect discrepancies between declared and actual items. As smart shopping carts offer cost-cutting potential for retailers and an improved shopping experience for consumers, the global market is projected to grow at a compound annual growth rate of 27.48%, reaching USD 9.74 billion by 2030 [4].

However, the grocery shopping experience remains time-consuming and frustrating due to several factors, many of which current smart cart solutions could address, yet often fail to resolve effectively. One persistent challenge affects consumers with dietary restrictions, such as allergies, who must frequently pause to examine ingredient lists and nutrition labels, significantly slowing their shopping process. Another common issue is the lack of route efficiency: consumers typically pick up products in a scattered way, resulting in unnecessary backtracking. This inefficiency becomes even more pronounced when consumers are unfamiliar with the supermarket layout or when certain products are either temporarily out of stock or simply not sold in that supermarket, leading to additional time spent trying to locate them. In such cases, when a product cannot be found, consumers often have to start a second search - this time for a staff member who can help. For some consumers, when one is finally found, the interaction itself may be uncomfortable or inconvenient [5]. Lastly, even after completing their shopping, consumers often realize they have forgotten to purchase essential items, leading to repeated visits to the supermarket.

An effective solution must aim to eliminate, or at least reduce, the likelihood of these factors arising during the shopping process. Addressing some of these challenges requires an advanced form of support that can adjust to consumer needs in real-time. This can be achieved through personalization, which has emerged as a key strategy for improving customer experiences across industries, particularly in retail environments. Personalization can be understood as the process of modifying a system's functionality, interface, content, or information access in order to make it more personally relevant to the user or a group of users [6]. Typically, personalization efforts have relied primarily on static data, such as purchase histories. However, recent advances in large language models (LLMs) have made it possible to create AI assistants that dynamically adapt system behavior by proactively engaging with users during interactions and understanding their evolving preferences [7].

Building on these advancements, this project proposes a smart cart system integrated with a digital companion, designed to provide on-demand, tailored product support through intuitive, voice-based interactions. In addition to the companion's AI-driven capabilities, the system also features complementary functions - such as route guidance and innovative recommender subsystems. Together, these capabilities aim to set a new standard for what smart retail technology can offer.

2 Literature Review

This chapter presents the general background necessary to understand the key concepts underlying the project. It covers both conceptual and technological foundations to clarify relevant terms, tools, and limitations.

2.1 Landscape of Dietary Needs

The modern dietary landscape is shaped by a complex interplay of health conditions, ethical values, and individual preferences, all of which influence consumer food choices and demand greater transparency.

2.1.1 Food Allergies

A food allergy (FA) is when a person's immune system overreacts to something in a certain food. This part of the food that the body reacts is usually a protein and is called an allergen. The immune system thinks the allergen is a threat and tries to attack it. This response can cause a range of symptoms, which are called an allergic reaction [8].

FAs represent a significant global public health challenge, with diverse prevalence rates across regions and demographics. In the United States alone, approximately 33 million individuals are affected by at least one food allergy. The most common food allergens in the US are peanuts, milk, tree nuts, eggs, wheat, soy and sesame [8].

The prevalence of FA has been consistently increasing for decades. Contributing factors include genetics, atopic diseases, family history, increased hygiene, and timing of food exposure. This escalation is often termed an "epidemic" [9].

2.1.2 Health and Ethical Diets

In recent years, a growing number of consumers have adopted special diets that reflect health-related needs. These dietary choices are needed to manage medical conditions and improve overall wellness.

Health-related dietary patterns include low-sugar, low-sodium, and low-fat which are commonly recommended for managing conditions such as diabetes, hypertension, obesity, and cardiovascular disease [10].
High-protein diets have gained popularity for weight management and muscle maintenance [11].

In addition to health concerns, many consumers adopt plant-based or reduced animal-derived diets which are often motivated by concerns about animal welfare, climate change, and sustainable food production. However, they are also frequently chosen for their perceived health benefits, such as lower intake of saturated fats and cholesterol.

Veganism and vegetarianism are two such dietary patterns that differ in their level of restriction. A vegetarian diet typically excludes meat, poultry, and fish, but may include dairy products and eggs. Individuals who follow a mostly vegetarian diet that includes fish are referred to as pescatarians. In contrast, a vegan diet excludes all animal-derived ingredients, including meat, dairy, eggs, and often honey [12].

2.1.3 Food Labeling

Food labeling presents information on packaged food products. These labels serve as the primary source of product information that consumers use to make purchase decisions aligning with their dietary needs.

Food labeling practices differ across countries due to variations in legal frameworks, cultural norms, and public health priorities. In most cases, core nutritional information, such as ingredients, allergen disclosures, and nutritional values, is legally required and typically displayed on the back or side of the packaging.

Some countries, including Israel, have introduced mandatory front-of-pack warning labels indicating high levels of sugar, sodium, or fat, as part of public health efforts to improve dietary choices. This approach is part of a growing global trend aimed at providing clearer nutritional information and encouraging healthier eating habits [13].

In addition, manufacturers sometimes voluntarily include front-of-pack labels, such as “vegan” or “gluten-free”, to appeal to consumer needs and preferences [14].

Despite ongoing efforts to communicate food quality, whether through voluntary labeling, when it serves the interests of manufacturers, or through mandatory regulations - understanding back-of-pack nutritional information has been found to be challenging for consumers that try to lead a healthy lifestyle and individuals with allergies. One major concern is that consumers state that they do not understand the provided information and that it is too small to read [15].

Figure 1. *A consumer reviewing a nutrition label, which is noted for its small font size and density.*

Additionally, since critical allergen information is not typically highlighted, consumers need to carefully review ingredient lists each time they shop, as product formulations may change and new allergens may be introduced [16].

2.2 The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is one of the most extensively studied problems, with broad applicability across various domains. It is named after the hypothetical scenario of a salesman who must travel a set of cities, visiting each exactly once, and returning to the starting point, while minimizing the total travel distance [17,18].

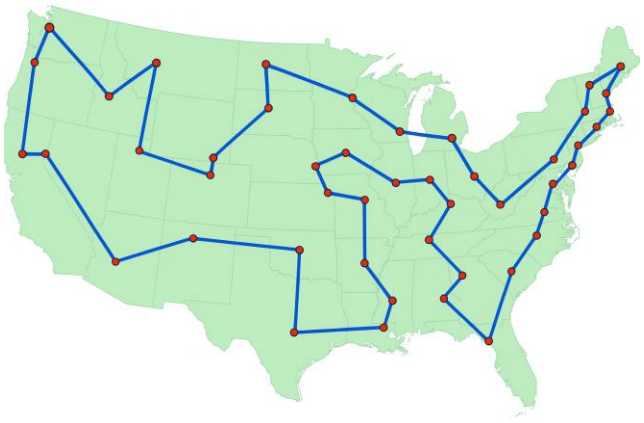

Figure 2. *Visualization of a TSP solution over cities in the United States.*

2.2.1 Importance and Applications

Academically, the TSP serves as a theoretical benchmark in computer science, as many algorithmic breakthroughs and heuristic frameworks were first tested or motivated by it.

Key applications of the TSP include logistics and supply chain management (such as optimizing delivery routes and transportation schedules), manufacturing (for example, determining the most efficient order of operations in a production line), microchip design (minimizing wire connections), and bioinformatics (for instance, arranging DNA fragments) [17,18].

Efficient solutions to the TSP directly translate into at least one of the following - depending on the specific application domain: reduced operational costs, improved processing times, optimized resource usage, or more accurate results. These outcomes underscore the practical relevance of the problem.

2.2.2 Formal Definition

Given a set of n nodes $V = \{v_1, v_2, \dots, v_n\}$ and a function $d(v_i, v_j)$ specifying the cost associated with the edge between nodes i and j, the objective is to find a permutation $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, where each $\pi_k \in V$ appears exactly once in the sequence, and the total cost $L(\pi)$ is minimal:

$$L(\pi) = d(\pi_n, \pi_1) + \sum_{i=1}^{n-1} d(\pi_i, \pi_{i+1})$$

The problem is typically classified as symmetric (STSP) if $d(v_i, v_j) = d(v_j, v_i)$, and asymmetric (ATSP) otherwise [18].

2.2.3 Computational Complexity and Implementation Challenges

In the ATSP, where the cost between two nodes depends on direction, the number of distinct solutions is $(n-1)!$, since we fix a starting node and consider all permutations of the remaining ones. In the symmetric case, where each sequence and its reverse are equivalent, this number is halved, resulting in $\frac{(n-1)!}{2}$ unique permutations. This makes a brute-force search for a minimal $L(\pi)$ impractical even for moderate n.

The challenge goes beyond brute-force infeasibility, as the TSP is an NP-hard problem, meaning no known algorithm can guarantee an optimal solution in polynomial time for all instances. As a result, the computational effort required to solve the problem optimally, grows rapidly with the number of nodes, making exact solutions infeasible for large instances.

As a result, developing algorithms for the TSP is challenging. Most approaches provide only near-optimal solutions (near-minimal $L(\pi)$), and even these require careful design to balance solution quality and runtime.

2.2.4 Algorithmic Solution Methodologies

The academic literature on TSP solution approaches can be categorized into exact algorithms, approximation algorithms (heuristics), and metaheuristics [18].

Exact algorithms guarantee finding the optimal solution, but their computational cost typically limits their applicability to instances of relatively small size. One such algorithm is dynamic programming, which systematically breaks the problem into smaller subproblems and stores their solutions to avoid redundant computations. Another approach is integer linear programming, which formulates the TSP as a set of linear equations and inequalities, with constraints that enforce the structure of a valid permutation.

Heuristic algorithms aim to find good solutions, though not necessarily optimal, in polynomial time, making them essential for large-scale instances where exact methods are computationally impractical. These include greedy strategies such as the nearest neighbor algorithm, which starts at a random node and repeatedly visits the closest unvisited node until all have been visited. Other approaches, like 2-opt and 3-opt local search, begin with an initial route, often generated by another simple heuristic, and attempt to improve it by removing and reconnecting edges in different ways to shorten the overall path.

Metaheuristics are high-level frameworks that guide heuristics in exploring the space of possible solutions more effectively. While basic heuristics often stop at a local optimum (a solution better than nearby alternatives but not necessarily the best overall), metaheuristics use strategies that continue the search for better solutions. Notable examples include genetic algorithms, which maintain a population of candidates and combine elements from the best ones to create improved candidates over iterations, and ant colony optimization, inspired by how real ants find short paths.

2.3 Large Language Models

Large Language Models (LLMs) are deep learning models trained primarily on large-scale natural language data, as well as other forms of text such as programming code and structured data like JSON, XML, and tabular datasets [19]. In some cases, extensions of LLMs are also trained on non-text modalities such as images and audio, resulting in multimodal models. Through this training, they learn to capture patterns, semantics, and contextual relationships within and across modalities. As such, they are able to generate responses across diverse inputs (called prompts) that give the impression of human-like understanding, making them useful in both interactive and task-oriented applications [20].

2.3.1 Hardware Requirements of LLMs

LLMs require high-performance hardware to operate effectively, particularly in real-time or large-scale interactive settings. Their ability to generate coherent and timely responses depends on access to sufficient memory bandwidth and parallel processing capabilities. Without such resources, performance may degrade due to increased latency, reduced throughput, or instability in longer sequences [21].

2.3.2 Operational Resource Demands

LLMs demand extensive computational resources not only during training but throughout their operational use. They have been shown to consume substantial energy and processing power, leading to increased financial cost. However, researchers have identified several methods, and are actively exploring additional approaches to reduce energy consumption [22].

2.3.3 Prompt Engineering

The foundation of LLMs has been made possible by the Transformer architecture. At its core is the self-attention mechanism, which, when applied to text, evaluates how each token, that typically represents a word, a sub-word or a symbol, in a sequence relates to every other token, regardless of their distance or order [23]. This mechanism enables the model to build rich contextual representations of the input, which are then used to probabilistically predict the next tokens in the output sequence [24].

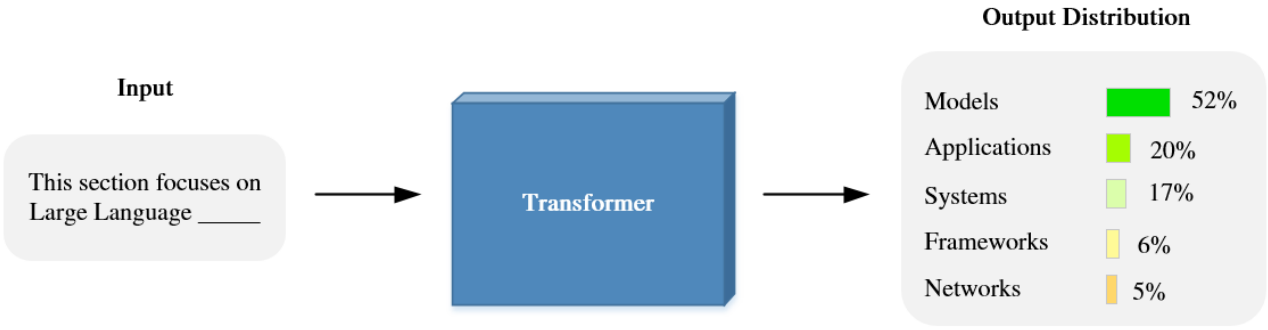

Figure 3. *Illustrative example of probabilistic next-token prediction using a Transformer.*

Since LLMs generate responses probabilistically, even small changes in the wording of a prompt can lead to significantly different outputs. This makes it crucial to craft carefully worded and precise prompts to guide the model toward more accurate and useful responses. The practice of doing so is known as prompt engineering.

The process of prompt engineering can be outlined in five general steps, which are not mandatory but serve as a useful guidance [25], outlined in Table 1 below.

Table 1. *Key steps in the prompt engineering process*

Step	Action	Description	Rationale
1	Define the goal	Specify the intended outcome of the prompt	Provides a clear direction for constructing the prompt
2	Analyze model capabilities	Assess the model's strengths and limitations	Determines whether the intended outcome is achievable given the model's capabilities
3	Structure the prompt	Organize the prompt using a clear and logical layout	Enhances the model's ability to interpret the input
4	Provide context	Include relevant background or situational information	Minimizes uncertainty in prompt interpretation
5	Refine the prompt	Test the prompt and revise it based on observed outputs	Enables optimization of the prompt

While these general steps offer a strong foundation, more sophisticated prompting strategies have emerged to further optimize LLM responses and reduce resource demands [25]. A selection of such advanced strategies is summarized in Table 2.

Table 2. *Examples of advanced prompting strategies*

Strategy	Description	Purpose
Few-shot Prompting	Includes few examples of questions and answers directly in the prompt	Helps the model learn the desired format, tone and level of detail
Role-based Prompting	Specifies the role the model should play	Ensures responses align with a defined persona
Chain-of-thought Prompting	Guides the model step-by-step to reach a conclusion	Enables responses in a coherent and logical manner
Instruction Prompting	Uses direct commands	Provides control over response characteristics

Beyond improving output quality, prompt engineering also plays a direct role in reducing the number of tokens consumed during interactions, which leads to lower computational costs [26]. This reduction is achieved in several ways:

1. The prompt is made more concise by avoiding irrelevant content and eliminating redundant instructions.
2. The prompt is structured to guide the model toward generating shorter responses, either through explicit instructions or by framing the task in a way that encourages brevity.
3. The response is more accurate and relevant, minimizing the need for additional interactions with the model.

2.4 Voice Assistants

Voice assistants are software agents that process human speech and respond using synthesized audio output, in order to answer questions or performing tasks. Since the early 2010s, their integration across services has been increasing, and IoT device manufacturers have also incorporated voice control into their products [27], as it offers users a hands-free and intuitive experience.

2.4.1 Voice Assistants Core Intelligence Transformation

The core intelligence driving voice assistants has undergone a significant transformation in recent years.

Early pioneers such as Apple's Siri and the first-generation of Amazon's Alexa primarily relied on rule-based frameworks. In these assistants, each well-defined task area (for instance Calendar or Messages) was packaged with a short list of if-then rules and the data those rules operate on. When a user spoke, the assistant first chose the right task area and then ran its rules to fulfil the request [28]. While revolutionary for their time, they struggled with complex or ambiguous inquiries, often resulting in constrained interactions that failed if an inquiry deviated from programmed rules. Moreover, they could not sustain multi-turn dialogues.

However, the advent of LLMs marked a profound paradigm shift, enabling their integration into modern voice assistants. They allow the systems to handle open-ended questions, engage in multi-turn dialogues, and perform complex reasoning, transforming them into fluid and capable conversational partners [29].

2.4.2 Speech-to-Text Process

The initial step in a voice assistant's interaction involves accurately converting speech into readable text. This crucial step is handled by a process known as Speech-to-Text (STT), or Automatic Speech Recognition (ASR).

The process begins with audio input, which is digitized and undergoes essential preprocessing to enhance clarity and standardize the sound. This processed audio is then transformed into numerical representations, that capture the sound's frequency and energy distribution.

These features are fed into an acoustic model, trained on large-scale audio-text pairs. The model analyzes the sound patterns and maps them to speech sounds or sub-word units. The output – probability distributions over possible sequences, is then passed to a decoder.

The decoder plays a crucial role in producing the most accurate transcription of the spoken input. To do so, it incorporates a language model, which has been trained on extensive text dataset. This model helps resolve ambiguities in the acoustic predictions (for example, distinguishing between “to” and “two” based on context) and ensures that the final transcribed text is both acoustically probable and grammatically correct [30].

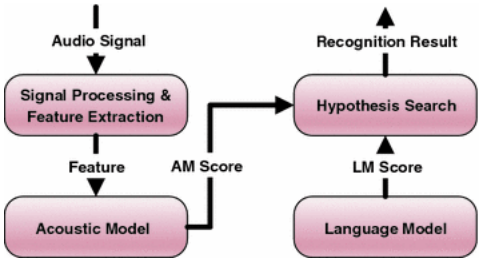

Figure 4. *Architecture of Speech-to-Text process.*

2.4.3 Text-to-Speech Process

The Text-to-Speech (TTS) process is responsible for converting the textual response given by the voice assistant’s core intelligence into synthesized, human-like speech.

It begins with text normalization, which expands abbreviations (for instance, “Dr.” becomes “Doctor”) and converts numerical expressions into words (for example, “123” becomes “one hundred twenty-three”).

The normalized text then undergoes text analysis, where it is transformed into linguistic features such as phonemes, stress, and rhythm. These features are passed to an acoustic model, which predicts corresponding acoustic features.

Finally, a vocoder uses these acoustic features to synthesize the actual audio waveform that the user hears [31].

Modern vocoders produce speech that sounds remarkably clear and natural, closely resembling a real human voice.

2.5 Recommender Systems

Recommender systems are software tools designed to suggest the most relevant items - such as products or services, to specific users, by estimating their preferences. These predictions are based on available data about the users, the items, and past interactions between them. They are widely used across various digital domains such as e-entertainment, e-commerce and e-tourism [32]. These systems not only help determine what to offer to an individual customer, but also enhance customer loyalty, as consumers tend to return to the platforms that best serve their needs [33].

2.5.1 Recommender Systems Major Challenges

Recommender systems introduce several challenges [33]. This section discusses two of the most significant.

One major challenge is producing personalized recommendations for new users, often referred to as the cold-start problem. This occurs when there is insufficient information about a user's preferences to generate relevant suggestions. A common solution is to collect some initial input from the user such as preferences.

Another key challenge is gaining user trust and acceptance on those systems. Users are more likely to accept recommendations when they understand why a particular thing was suggested. A lack of transparency can lead to skepticism. Therefore, clarifications about suggestions are required.

2.5.2 Recommendation Techniques

Understanding different recommendation techniques is essential for grasping how recommender systems function. Table 3 presents several widely used techniques, along with the contexts in which each is typically applied [32,34].

Table 3. *Comparison of recommendation techniques*

Technique	Core Idea	Suitable When
Content-Based	Recommendations of items similar to those the user liked before	Personalized recommendations are needed and user-specific preferences are available
Collaborative Filtering	Recommendations based on users with similar preferences	Patterns across users are valuable and user-item interaction data is rich
Knowledge-Based	Rule-based recommendations that match items to explicit user needs	Decisions are goal-driven and historical data is irrelevant
Context-Aware	Recommendations based on situational context	The user's preferences vary depending on contextual factors
Hybrid	Combines two or more techniques	Aiming to improve accuracy or overcome limitations of a single method

3 Overview of Smart Cart Solutions

This chapter presents an overview of selected smart cart solutions that have been deployed in Israel. Examining these systems helps identify common capabilities and design patterns. While the focus is on systems in local use, the technological features they offer and their structural design are largely consistent with smart cart solutions found globally.

Detailed technical documentation on commercial smart cart systems is limited. Therefore, the analysis presented in this chapter is based mostly on promotional materials, demonstrations, and deployment case studies provided by the companies developing these systems.

3.1 Supersmart XP

The Supersmart XP solution offers a flexible smart cart platform that minimizes the need for physical modifications to shopping carts. It operates either as a mobile application – compatible with Android and iOS smartphones using computer vision for item scanning, or via a dedicated handheld device, the Zebra PS20.

The PS20 is based on a system-on-chip (SoC) architecture that supports the Android operating system, which enables application to run directly on the device. The device integrates a 4-inch touchscreen, a barcode scanner, a microphone and a speaker. Its high-efficiency battery supports fast charging, providing approximately 45 minutes of operation from just 15 minutes of charging [36].

Figure 5. *Zebra PS20 handheld devices in use at Osher-Ad.*

The platform delivers product recommendations [37], by suggesting additional units of the scanned items to qualify for a discount, and centralized analytics tools for retailers. Since it does not incorporate weight sensors, all items sold by weight, such as fruits, vegetables, meat, fish, and cheese must be pre-weighed at dedicated stations, where a barcode sticker is printed and then scanned.

For checkout, the system incorporates a walk-through validation unit that combines weight sensors and cameras to verify the contents of the cart automatically [38].

According to Supersmart, its deployment at Osher-Ad has yielded substantial operational improvements, including a reduction in shopping time from 90 to 40 minutes, the elimination of checkout queues, increased average basket value and a decrease in cart abandonment rates from 10% to less than 1% [37].

3.2 A2Z Cust2Mate

The earlier version of the Cust2Mate smart cart system is a fully integrated solution, in which the smart components are embedded directly into the body of the cart. The system includes a 13.3-inch touchscreen, an integrated barcode scanner, a weighing platform, and an on-cart payment terminal enabling the entire shopping process to be completed directly on the cart. Upon collecting the cart, users can be easily identified by entering their phone number [39].

Figure 6. *Cust2Mate system integrated into a cart at Yochananof.*

According to A2Z, the deployment of this version of smart carts at Yochananof has led to 73% returning customer rate and a 70% increase in average basket size [40].

In its latest iteration, Cust2Mate has introduced a modular clip-on version, where the smart panel is now detachable. The updated system introduces new features, including computer vision-based loss prevention, synchronized shopping list access, and a product search tool that displays the item's location on a store map [41].

3.3 Shopic Shop-E

Shop-E is a clip-on device that attaches to standard shopping carts and includes a touchscreen, a barcode scanner and two cameras.

Figure 7. *Shop-E clip-on device in use at Shufersal.*

In contrast to Cust2Mate, which primarily uses computer vision to reduce fraud, Shop-E leverages computer vision to automatically identify most products as they are placed into the cart, reducing the need for manual barcode scans [42]. As for weighted items, just like in the Supersmart XP system, they are weighed separately and labeled with a barcode.

Shop-E offers a recommender system that promotes products based on the location where the customer picked a previous item. It also allows customers to call staff for assistance directly from the cart interface, as well as pay either through a smartphone application or at a self-service kiosk using a credit card.

3.4 Comparative Summary

The following table summarizes the core aspects of the smart cart systems discussed in this chapter. It highlights physical design, deployment characteristics, and technical capabilities, offering a clearer understanding of their similarities and differences.

Table 4. *Comparative overview of core aspects in selected smart cart systems*

Aspect	Description	Supersmart XP	Cust2Mate	Shop-E
Physical Design	How the unit is attached to the cart	Handheld device	Fully integrated; Clip-on device (latest version)	Clip-on device
Display Type	Type of screen used	Touchscreen	Touchscreen	Touchscreen
Display Size	Diagonal screen size	4-inch	13.3-inch	*Unknown*
Operating System	The software that runs the system's application	Android	*Unknown*	*Unknown*
Charging Station	Infrastructure used to store and charge units	Dock	Cart rail; Dock (latest version)	Dock
User Authentication	How the user logs in	ID number	OTP via SMS	ID number
Item Recognition	How items are added to the cart list	Manually	Manually	Automatically (non-weighed items)
Item Recognition Interface	Hardware used to add items	Barcode scanner	Barcode scanner	Cameras, Barcode scanner
On-Cart Item Weighing	Availability of an integrated weighing mechanism	No	Yes	No
Item Weighing Interface	Hardware or external unit used to weigh items	Barcode-generating weigh station	Integrated scale	Barcode-generating weigh station

Note. OTP stands for One-Time Password (or One-Time PIN), which is valid for only one login session.

Table 4 *(Continued)*

Aspect	Description	Supersmart XP	Cust2Mate	Shop-E
On-Cart Payment	Capability to complete payment directly at the cart	No	Yes	Yes
Payment Interface	How the checkout process is completed	Walk-through validation unit	Integrated terminal	Smartphone application / self-service kiosk
Recommender System	Type of product suggestions provided	Quantity-based promotions	Personalized promotions	Location-based promotions
Key Features	Notable additional capabilities	Analytics tools for retailers	Product locator via search; Synchronized shopping list (Both in the latest version)	Call staff for assistance
Estimated Engineering Cost Per Unit [57]	Approximate total development cost per unit encompassing hardware, software, QA efforts and mechanical design	USD 800 (± 15%)	USD 2100 (± 20%)	USD 900 (± 15%)

Note. The engineering cost estimates were generated using GPT-3.5, based on the specified aspects, device images, and publicly available information retrieved during the interaction.

3.5 Observations and Discussion

This section outlines insights based on the summary, which can help inform the development of a more innovative smart cart solution aimed at addressing the factors that make the shopping process time-consuming.

3.5.1 Evaluation of Physical Design

Fully integrated units, such as the earlier version of Cust2Mate, are more difficult to maintain, increase the cart's weight and, if not waterproof, cannot be taken outside the supermarket, which requires customers to transfer their items to a regular cart after checkout. These drawbacks likely contributed to Cust2Mate's transition to a modular clip-on system in newer models, allowing the smart unit to be detached.

3.5.2 Evaluation of Computer Vision-Based Scanning

The detection accuracy is often insufficient, leading to frequent recognition errors and disruptive UI warnings. As a result, a barcode-based approach may offer a more reliable and user-friendly experience.

3.5.3 User Identification and Personalization

Delivering personalized promotions and recommendations requires accurate user identification. Cust2Mate's use of OTP-based authentication rather than just entering an ID number likely reflects an effort to ensure the user's identity.

3.5.4 Effectiveness of Key Features in Practice

While features like staff assistance, product location, and shopping list synchronization appear to address challenges outlined in the introduction, real-world usage reveals several usability issues that limit their effectiveness. According to reviews by consumers, the button to call staff for assistance fails as the staff does not even appear. Additionally, the product locator requires typing the product name, which is not ideal for people who struggle with that and therefore they might not use it. Lastly, synchronized lists merely appear statically on the screen, without any guidance, thus maintaining a scattered pick-up order pattern.

4 System Characterization

This chapter details the core characteristics of the proposed solution. It begins with a description of the unique capabilities and the specific problems each one addresses and solves, in addition to the baseline capabilities typically found in any smart cart, such as product scanning and tracking the total cost during shopping. It then provides an overview of the benefits for key stakeholders, including consumers, retailers, and manufacturers. This is followed by a breakdown of the system's hardware and software components, describing the role of each element and its necessity in enabling the smart cart's full range of capabilities. Finally, the chapter concludes with a basic flow diagram that captures the general sequence of actions, to aid overall understanding.

4.1 Unique System Capabilities

4.1.1 Interactive Nutritional Guidance

The system offers real-time guidance to help users in understanding and evaluating the products they encounter during shopping with an integrated digital companion. Upon scanning a product, the digital companion can answer user inquiries related to nutritional content, allergen presence and ethical alignment. This capability is particularly valuable for users who find it difficult to interpret ingredient lists or read them due to small font size and dense information, enabling them to determine whether a product suitable for their dietary needs and restrictions.

4.1.2 Personalized Warnings

The system dynamically generates personalized warnings based on user-specific health or ethical preferences, inferred through follow-up questions posed by the digital companion immediately after inquiries about nutritional information. When a product is scanned, it will be flagged if it conflicts with any of the user's stated concerns, directly in the user interface. These visual alerts are designed to assist the user by reducing the effort associated with manual label inspection.

4.1.3 Recommender Subsystems

The system employs two distinct recommender subsystems: alternative product suggestions module and a frequently-bought products suggestions module.

The alternative product suggestions module combines a knowledge-based approach enhanced with contextual user data. It operates in two modes: proactively, by automatically recommending suitable alternatives when a scanned product conflicts with the user's inferred restrictions, leveraging structured knowledge such as dietary indicators; and reactively, through interaction with the digital companion, which responds to explicit user requests for alternatives - sometimes involving complexity, by comparing the nutritional data of the most recently scanned product with others in the same category, and providing explanation for the suggestions. This reduces the need for manual search of products when seeking compliant substitutes.

The frequently-bought products suggestions module leverages a content-based filtering approach. By analyzing prior purchase data, the system identifies habitually selected items that are missing from the current shopping

session. During checkout, it suggests these potentially forgotten products, thereby helping to reduce the risk for the need to visit the supermarket again to purchase them.

4.1.4 Product Location Guidance with Availability Awareness

As part of the digital companion's assistance capabilities, the system provides real-time guidance on product locations through a visual representation of the supermarket layout. Unlike other smart cart systems that display static shelf locations, this feature is designed not only to assist newcomers unfamiliar with the layout but also to benefit users who are already familiar with the supermarket's product placement, by incorporating stock awareness. The companion verifies product availability in real time and, if an item is out of stock, immediately alerts the user, thereby preventing unnecessary trips to empty shelves. Consequently, this functionality reduces wasted time and minimizes the need to seek assistance from an employee.

4.1.5 Route Optimization with a Pre-Created Product List

The system supports route optimization for product collection. Through an interface accessible from a personal device, users can create and save their shopping list in advance. Upon arrival at the supermarket, the user can simply load the pre-defined list into the smart cart with the press of a button. The digital companion then displays a visual map of the supermarket, with numbered markers placed directly on the product locations to indicate the recommended pickup order for an efficient route through the supermarket. As finding good order to pick up items is equivalent to a TSP problem, a TSP solver is required to find a near-optimal pickup order within a reasonable time. This feature can minimize the shopping time, as it may reduce unnecessary walking.

4.1.6 Voice-Based Interactions

Interaction with the digital companion is done primarily through voice using natural language, allowing users to communicate with the system in an intuitive and accessible way. This is enabled by the integration of AI models - specifically STT, LLM and TTS. Compared to text-based input, voice interaction simplifies the process and allows users to use the companion more quickly, especially those who may find typing slow or challenging. This interaction reduces reliance on manual menu navigation typically required in traditional smart cart systems, and makes the system more convenient for people who are less comfortable with technology. To support usability in noisy or interruptive environments and help users keep track of the conversation, all voice interactions - both user inquiries and companion responses, are displayed in a chat panel. The interaction is initiated and terminated via a button located on the same panel.

4.2 Stakeholders Benefits

4.2.1 Consumers

As the primary users, consumers benefit not only from a more efficient shopping experience but also from a fundamentally more informed one, promoting long-term behavioral change toward healthier and more mindful consumption.

4.2.2 Retailers

As those responsible for shaping the consumer experience, retailers stand to benefit significantly from the integration of such a smart cart system, as it offers two key avenues for increasing revenue.

First, as the smart cart enables more satisfying and efficient shopping journey, it can attract a broader customer base - translating to more foot traffic and potentially higher overall sales.

Second, the recommender subsystems play a strategic role by reducing product abandonment. When a product does not meet the shopper's needs or preferences, the system suggests suitable alternatives, ensuring that the purchase opportunity is retained. This increases the likelihood of minimizing product drop-off rates and, therefore, boosting sales.

4.2.3 Manufacturers and Brands

Food manufacturers and brands can leverage the smart cart system as an opportunity to remain competitive and adapt to shifting consumer behavior. Products that meet a wider range of user preferences may gain increased visibility through the alternative product suggestions module, creating new opportunities for brand distinction and expanded market reach.

4.3 Hardware Components

The smart cart system integrates several hardware components that collectively support its full range of capabilities.

4.3.1 Computing Unit

The computing unit acts as the controller of the smart cart by managing peripheral hardware, running the operating system, and maintaining communication with remote servers.

4.3.2 Display

The display is used to present the graphical user interface. To enable intuitive operation of the system, as in typical smart cart systems, the display supports capacitive touch, since using a keyboard and mouse is impractical in this context. It is also large enough to comfortably present visual elements such as the supermarket map and interaction history.

4.3.3 Barcode Scanner

The barcode scanner is an essential component in typical smart cart systems, enabling real-time identification of products as they are added to the cart. It is preferred over computer vision due to its greater reliability.

4.3.4 Audio Interface

The microphone and speakers are essential for enabling voice-based interaction with the digital companion. The microphone captures the user's speech, while the speakers deliver the companion's responses for the user to hear.

4.3.5 Battery

The system is mobile and therefore requires a rechargeable battery. The battery is designed to support the entire duration of a shopping session, ensuring continuous operation without unexpected shutdowns.

4.4 Software Components

A combination of software components powers the system's functionality.

4.4.1 Operating System

The operating system provides the foundational environment for running the smart cart's software stack. It manages hardware resources, hosts the high-resolution graphical user interface, and handles low-level communication with peripheral hardware.

4.4.2 User Interface

The UI guides users through the shopping process and presents relevant product information. It runs in kiosk mode, restricting access to intended functions only. Designed to be responsive and lightweight, it adapts to various screen sizes and performs reliably on limited hardware. To ensure consistency across carts and reduce maintenance overhead, the interface must support streamlined deployment and remote updates.

4.4.3 AI and Backend Services

As mentioned, interaction with the digital companion relies primarily on the integration of AI models that together form a voice-based assistant. At its core is an LLM, responsible for interpreting user intent and generating natural language responses. These responses may be based solely on the model's pre-trained knowledge, including nutrition-related information, or enriched through integration with backend services that retrieve product-specific data from the system's database, ensuring accurate and context-aware replies. In some cases, the LLM also triggers backend services to support specific capabilities such as the alternative product suggestions module or product location guidance. These services are built to be scalable, supporting concurrent usage by multiple smart carts while maintaining low latency and reliable operation.

4.4.4 Authentication Service

The authentication service is used during registration and login and is necessary to personalize the smart cart experience based on the user's profile. The user provides a phone number and receives an OTP via SMS. This approach enables convenient user verification without relying on usernames or easily forgotten passwords. It also adds a layer of trust by helping ensure the person accessing the system is truly the owner of the registered identity, unlike static identifiers such as an ID number that can be entered by anyone.

4.4.5 Database

A database is used to store user information, such as preferences and shopping history, as well as product information, including ingredients and availability. It provides fast query performance to ensure a responsive user experience, especially during product scanning.

4.5 System Flow

Following the characterization, a simplified overview of the smart cart system's operational flow during a typical shopping session is presented below. It highlights the main user actions and system responses.

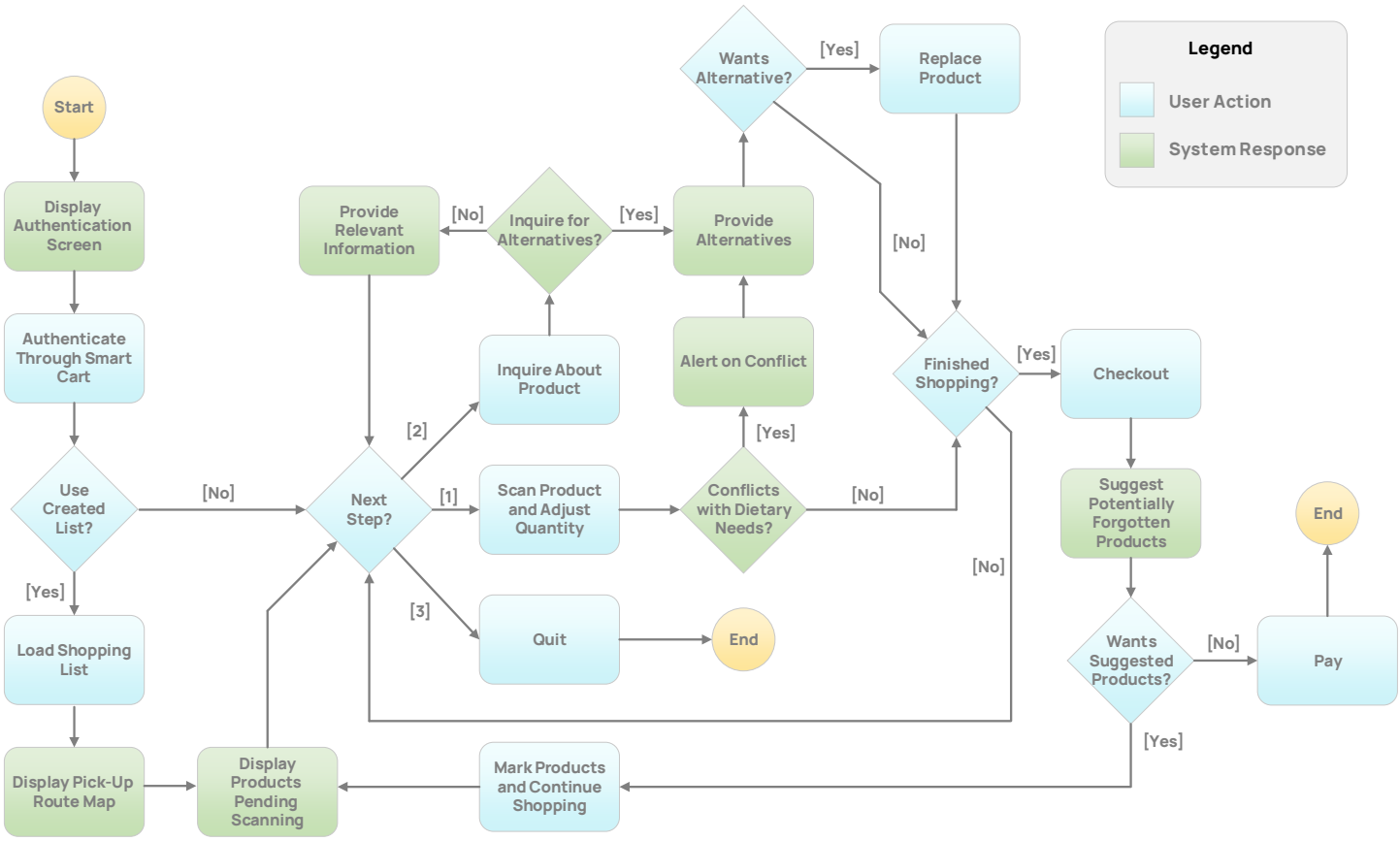

Figure 8. *High-level illustration of the smart cart system flow.*

4.6 Enabling Technology

This solution relies on the integration of an LLM with backend services, which raises questions regarding the feasibility of such an approach.

Fortunately, recent advancements in LLMs have enabled them to function as autonomous agents, capable of invoking external tools or services when needed. One of the technologies that enables this capability is known as function calling. In this method, a set of callable functions are explicitly defined, including their names, parameters, and descriptions. These definitions are sent with each prompt to the LLM. Based on the input and the available functions, the LLM may decide to invoke one of them. The system then executes the selected function and returns the result to the model, which incorporates the output into its final response [43,44].

The following diagram illustrates this process and how function calling enables the LLM to interact with system components.

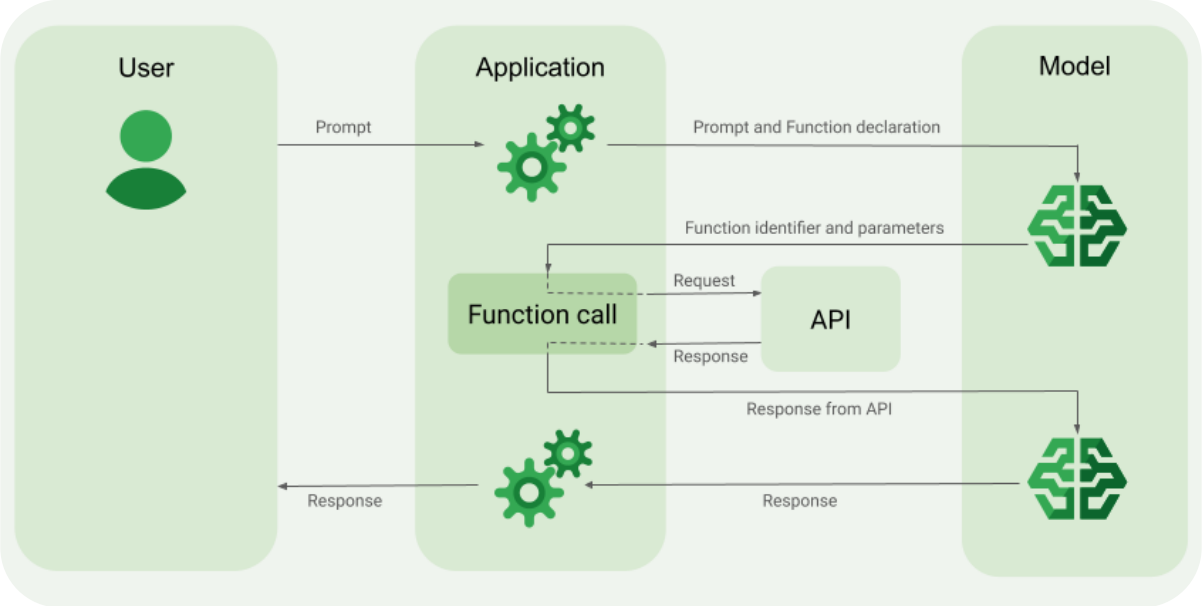

Figure 9. *Function calling process.*

5 Expected Achievements

5.1 Outcomes

If successfully implemented, the smart cart system will achieve two key outcomes. The primary outcome is that it will demonstrate a more efficient and interactive shopping experience for supermarket customers, as detailed in the system characterization. But beyond that, it will establish the feasibility of extending such capabilities to a broader range of products and retail scenarios.

Within the supermarket setting, while the solution proposed in this project focuses on delivering more informative experiences related to food products, it could also be adapted to support non-food items, answering user inquiries about product characteristics that reflect personal values or ethical concerns - such as environmental sustainability, recyclability, or animal testing. This would transform the assistant into a general-purpose consumer advisor that promotes responsible and conscious purchasing decisions beyond dietary considerations.

Furthermore, the underlying architecture, centered on real-time, natural language interaction - holds significant potential for broader commercial applications outside of supermarkets. For example, in restaurants or drive-throughs, the same conversational interface could serve as a smart menu, allowing customers to inquire about ingredients, allergens, or preparation methods, and to place orders directly through the system without sacrificing convenience. In large-format retail environments such as furniture or home improvement stores, the assistant could help customers explore item suitability, compatibility, or usage guidance, reducing the need for staff assistance.

5.2 Success Criteria

5.2.1 Cost Feasibility

The system can be considered successful if it demonstrates the ability to maintain realistic operational cost. While actual deployment costs cannot be fully assessed at this stage, given that a significant portion of the usage cost could come from the voice-based assistant, the system will be evaluated based on a **demonstration of the ability to reduce the costs through design choices**.

5.2.2 System Usability

Like any system intended for customer use, the smart cart must be intuitive and easy to operate. Otherwise, it risks being rejected by users, resulting in wasted investment. A **score above the System Usability Score (SUS) average of 68** will be considered an indicator of success for this criterion.

5.2.3 Real-Time Performance

This criterion is particularly important in systems with an integrated voice assistant to maintain user trust, as significant delays may be perceived as a system malfunction and undermine the assistant's usefulness in reducing shopping time. Therefore, the system will be considered well-performing if the **companion initiates its response within 3 seconds following the completion of the user's inquiry**.

5.2.4 Responses Accuracy

While perfect response accuracy may be achievable under deterministic conditions, the use of an AI-driven companion introduces practical challenges such as ambiguous user phrasing and speech recognition errors. Consequently, the goal is to **achieve the intended response in at least 90% of user inquiries**.

5.2.5 System Stability

In actual deployment, smart cart systems are expected to operate stably under concurrent usage by many customers, particularly when integrating AI models that trigger backend modules. At this stage, a simplified benchmark will be used. The system will be considered **stable if it can successfully handle at least 3 parallel user inquiries without crashing**.

6 Engineering Process

The engineering process started with a review of existing smart cart technologies, to understand what already works and what's still missing. Based on that, a system characterization was proposed for a more personalized smart cart that integrates a digital companion designed to make the shopping experience more time-efficient, more intuitive, and for those with dietary needs – more informed and protected.

Keeping in mind that the architecture must align with the success criteria while also laying the groundwork for future deployment, the next step is to dive into detailed technical and architectural considerations - justifying each technology and design choice, while also explaining how the system will function.

6.1 Constraints and Technical Challenges

To begin the design process effectively, it is essential to first understand the limitations that set the boundaries of the project. Recognizing these constraints at an early stage enables informed decision-making and ensures that the research efforts remain focused.

6.1.1 Limitations Preventing Domain-Specific LLM Training

The digital companion in our system is designed to provide real-time guidance on supermarket products, including nutritional information, ingredient inquiries, and recommendations for alternatives. Ideally, this functionality would be powered by an LLM that has been fine-tuned on domain-specific datasets, such as scientific publications in nutrition, dietary, and health research. Such fine-tuning would enable the model to generate more accurate, reliable, and contextually relevant responses when addressing inquiries, thereby ensure the system stays within safe operational boundaries and avoids delivering potentially inaccurate advice.

However, fine-tuning a large-scale LLM presents significant challenges. This process requires substantial computational resources, specialized expertise in machine learning, and the collection of high-quality data in the nutrition and health domain. Moreover, the training process itself is both time-consuming and cost-intensive. Due to these constraints, the project must rely on a general-purpose LLM, which is trained on broad, publicly available data that may occasionally include inaccurate or unverified information.

To address the risks associated with using a general-purpose model - particularly in a domain where misinformation can have health implications, the LLM must be restricted in scope to provide informational support about products rather than offering dietary advice related to what is better for individual health. This can be achieved through prompt engineering techniques designed to constrain the model's behavior, in combination with access to the system's database to retrieve verified product information, such as allergen content or ingredient lists, thereby reducing dependence on the model's internal knowledge, which may be incomplete, outdated, or unverified. This strategy is commonly adopted in other LLM-driven frameworks [35].

6.1.2 Limitations Affecting Implementation Scale

Unlike commercial development teams with an extensive workforce, this academic capstone project is carried out by a small student team, within a fixed timeline and without the benefit of mechanical engineering expertise. As a result, it is not feasible to develop a complete, production-level smart cart system.

Therefore, physical cart integration and certain capabilities such as integrated payment processing, product weighing, real-time battery percentage monitoring and management interface for retailers, are excluded from the implementation scope. This decision stems from the fact that such capabilities are already widely implemented in existing smart cart solutions, whereas this project aims to demonstrate the practical application of new system capabilities within the software engineering domain.

To avoid redundant development effort the project leverages existing software components, rather than developing these complex components from scratch. This approach reflects standard industry practice, where proven tools are adopted to accelerate implementation.

6.1.3 Limitations in Resources and Infrastructure

As an academic project without commercial funding, the system is subject to budgetary limitations that influence decisions regarding the hardware and software architecture.

On the hardware side, cost constraints affect the selection of components and thereby the system performance and reliability. For example, the accuracy of voice-based interactions in noisy environments could be improved by using advanced microphones with noise-canceling capabilities. However, such equipment is often expensive.

On the software side, the system is planned to be deployed using platforms that offer free usage tiers. While these platforms support initial deployment and testing, they come with limitations in terms of scalability, availability, and access to dedicated resources. This stands in contrast to commercial-grade cloud infrastructures, which offer advanced capabilities such as auto-scaling.

6.2 Computing Platform Considerations

Since different computing platforms offer varying capabilities, it is important to analyze them, as not all can fulfill the operational demands defined in the system characterization.

In the context of IoT Applications, microcontrollers and single-board computers represent the two primary classes of computing units.

While both microcontrollers and single-board computers are self-contained computing units, they differ significantly in architecture, purpose, and capabilities. Microcontrollers are optimized for specific control tasks within embedded systems, offering low power consumption and cost-effectiveness. In contrast, single-board computers provide the functionality of a full-fledged computer, capable of running complex operating systems and handling general-purpose computing tasks [45].

To illustrate the difference between these two classes, the following comparison contrasts the ESP32, a widely used microcontroller, with the Raspberry Pi 5, a single-board computer.

Table 5. *Comparison between ESP32 and Raspberry Pi 5*

	ESP32 [46]	**Raspberry Pi 5** [47,48]
Computational Capability	Lightweight, single-purpose embedded tasks	Multitasking, GUI rendering, and full system management
OS Support	FreeRTOS	Linux
RAM	0.5 MB (with extended 8MB)	2 - 16 GB
External Storage	Up to 16 GB	Up to 1 TB
GPU	*N/A*	VideoCore VII
Touchscreen Support	Low-resolution SPI/I2C Display	High-resolution HDMI Display
Audio Capabilities	Low-level I2S Interfaces	Plug-and-play USB/HDMI Ports
Power Consumption	0.5 – 1 W	5 – 12 W
Communication	Wi-Fi, Bluetooth	Wi-Fi, Bluetooth, Ethernet
Cost	USD 10 - 20	USD 50 - 100

Taking into account this comparison, it is evident that an SBC is more suitable for the proposed system. Its ability to support a high-resolution touchscreen and run a modern operating system allows the integration of a graphical application along with the use of standard software frameworks, as seen in typical smart cart systems.

For these reasons, the system will rely on Raspberry Pi 5 with 4GB RAM. While industrial-grade SBCs such as the NVIDIA Jetson Nano offer enhanced performance and durability, they come at a significantly higher cost. The Raspberry Pi 5, by contrast, provides a well-balanced tradeoff between performance and affordability, making it a practical choice for an academic setting that aligns with the system needs.

Despite its suitability, the Raspberry Pi 5 presents several limitations that must be considered. First, although its power consumption is lower than that of a desktop computer, it is still significantly high. As a result, the system requires a sufficiently large and reliable battery to ensure uninterrupted operation throughout a shopping session. Moreover, due to the limitations of its CPU and lack of dedicated AI acceleration hardware, the Raspberry Pi 5 is not capable of running AI models locally without incurring noticeable latency. To maintain a responsive user experience, these models must be executed on remote servers.

6.3 Operating System Considerations

While Android - an operating system based on the Linux kernel and capable of running on the Raspberry Pi, is often considered suitable for touchscreen-based systems due to its interface optimization, the system will instead use Raspberry Pi OS, the officially supported operating system, for several key reasons. First, peripheral access in Android is more restricted and often requires additional abstraction layers or custom drivers, which can increase development complexity and time. Second, Raspberry Pi OS is significantly lighter than Android, consuming fewer system resources such as RAM and CPU. Lastly, it benefits from extensive official documentation and strong community support, which facilitates development and troubleshooting.

Nonetheless, Raspberry Pi OS is still capable of delivering a smooth touchscreen experience when paired with an appropriate UI framework and proper system configurations, as will be discussed in Section 6.5.1.

To run the operating system, the Raspberry Pi requires a microSD card, which serves as its primary storage. A card of at least 16 GB is recommended to ensure sufficient space for the system and updates. The installation is typically done using the Raspberry Pi Imager, an official utility that allows users to easily select and flash the desired OS image onto the card. Once flashed, the card is inserted into the Raspberry Pi, enabling it to boot into the operating system [49].

As a full operating system, Raspberry Pi OS requires periodic software updates to maintain security and stability. This is an important consideration in production-level deployments, as it introduces maintenance overhead. However, a simple automated shell script - using the Advanced Package Tool (APT), as recommended by the Raspberry Pi Foundation [50], can be configured to install updates during off-hours when the supermarket is closed. This approach minimizes disruption and helps streamline long-term system support.

6.4 External APIs Considerations

6.4.1 Voice Assistant

Integrating STT, LLM, and TTS into a voice assistant introduces complexity. Each component requires significant processing power and memory. As a result, such system must typically run on robust server infrastructure and demands a substantial engineering effort, often spanning weeks or even months, to optimize latency and ensure real-time performance, scalability, and responsiveness.

As discussed in Section 6.1.2, dedicating the majority of development resources to building and optimizing the voice interaction pipeline is not feasible within the available timeframe. Therefore, an external SDK that includes an API, referred to as VAPI, will be utilized in this project.

VAPI is a platform for building and deploying voice AI agents. Its SDK provides a high-level interface for orchestrating the stages of the voice interaction pipeline. It supports a wide selection of models for each stage and also allows developers to connect their own model servers, for more control. Developers can choose components based on functional needs, with real-time visibility into estimated latency and cost.

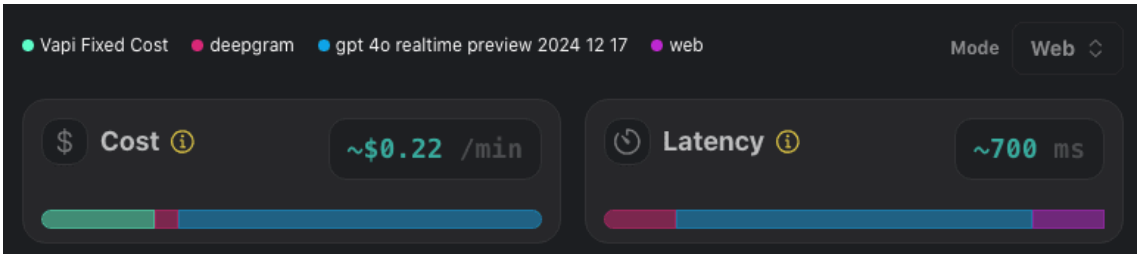

Figure 10. *VAPI interface displaying estimated cost and latency for selected pipeline components.*

The SDK offers configurable parameters for speech synthesis, including voice, pitch, rate, and language, and supports prompt engineering as well as the definition of function-calling endpoints to extend the assistant's capabilities. VAPI also enables uploading external files, typically used to provide contextual information to the assistant, and includes a dashboard for monitoring system performance, usage, and cost metrics. In addition, the platform offers testing tools that allow developers to evaluate and refine the assistant's behavior before deployment [51].

As this platform offers notable advantages in terms of ease of system integration, flexibility in models' selection, and transparency in cost and latency estimates, it is a suitable choice for this project.

Moreover, the ability to connect custom model server provides the option to pre-process user transcriptions before passing them to the language model - a possibility that is kept in consideration in case additional cost reduction techniques are required. This can involve filtering out casual conversational elements such as greetings, social pleasantries, or other inputs that do not require the core capabilities of the language model, in order to minimize token usage.

As model selection is required for each stage of the pipeline, selecting the appropriate models will be a key step in the system's development to ensure optimal results. Current comparisons (see Figure 11) suggest that Gemini 2.5 Flash represents the most balanced option for the language model component, the core of the pipeline, offering a strong combination of reasoning capability, fast response time, and low cost per token [52].

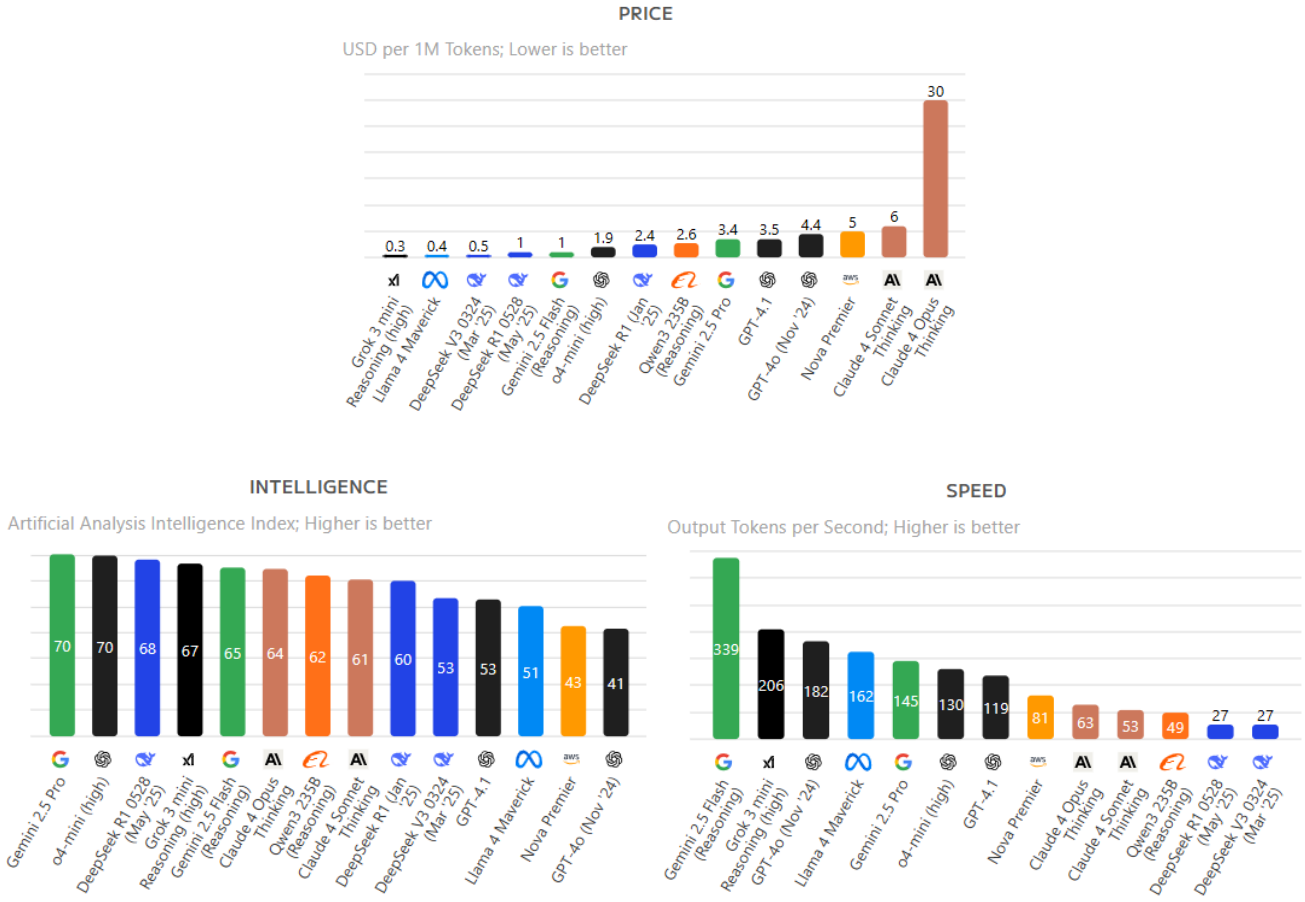

Figure 11. *Comparison of language models by price, intelligence and speed.*

However, given the rapid advancement in the development of AI models, the specific models to be used in the system have not yet been determined and will be selected based on future evaluations and testing.

6.4.2 OTP Verification

Implementing OTP-based verification typically relies on third-party platforms. These established platforms handle global delivery across multiple channels, controlling how many verification messages can be sent per time period to prevent spam and manage costs, and detecting and blocking attempts to exploit the OTP system.

As stated in the system characterization, the system is intended to use SMS-based OTP for authentication. This authentication flow operates through a coordinated sequence involving the frontend, backend, and third-party messaging service. When users enter their phone number, the frontend triggers a backend request that generates an OTP and sends it via SMS using the external messaging API. After receiving the code, the user submits it for verification. The backend then validates the OTP against the stored value and checks whether it is still within the valid time window. If the verification succeeds, the system queries the database to determine whether the phone number corresponds to an existing user. Returning users are granted access, while first-time users are first assigned a new record in the database - effectively eliminating the need for a separate registration process.

Currently, Twilio, a leading platform for communication services, is being considered for implementing the OTP verification via SMS, primarily due to its accessible free trial. The trial includes a small amount of credit and allows sending a limited number of messages to manually verified phone numbers - making it well-suited for prototyping and sufficient for demonstrating the registration and login flows. In addition, Twilio provides well-documented APIs and SDKs, enabling quick and reliable integration into the system.

As a cost-effective alternative, and given that the smart cart already utilizes a barcode scanner, generating a QR code on the user's smartphone for authentication is also being considered. However, this method may not be suitable for all users, particularly those who do not use smartphones

6.5 UI Considerations

6.5.1 Cart UI Implementation and Integration Strategy

Given the need for an interface that supports streamlined deployment and remote updates across all carts, a web-based user interface emerged as the most practical approach. Unlike native applications, which require updates to be applied on each individual device, a web-based solution can be maintained centrally, reducing maintenance effort and ensuring consistent behavior.

Among the various web technologies evaluated, all are capable of adjusting to different screen sizes and are lightweight. However, React is selected for its broader ecosystem support and widespread industry adoption, both of which provide confidence in the system’s long-term stability and maintainability.

React is a JavaScript library that enables development of user interfaces by composing them from modular pieces called components. This component-based architecture promotes code reusability. In addition, React uses a declarative approach, meaning that instead of writing step-by-step instructions for how the interface should change, developers specify what the interface should display based on its current state [53]. This makes the code more predictable, easier to debug, and easier to understand as the application grows.

Supporting a web-based interface requires a reliable browser capable of full-screen operation and robust touchscreen functionality. Chromium is chosen for its strong compatibility with Raspberry Pi OS and its support with handling touch interactions, which is essential in this kind of system.

For the interface to launch automatically and appear as part of a single-purpose system, Chromium is started at boot using a systemd service. This service manager, standard in Linux-based operating systems, controls process startup and enforces recovery policies [54]. Within this setup, Chromium would be launched in full-screen kiosk mode, automatically navigating to the designated URL, with additional flags to optimize touch responsiveness - achieving a user experience comparable to native Android interfaces. Kiosk mode would also play a key role in preventing access to browser controls, ensuring users remain within the application.

Since browsers may crash or become unresponsive, the systemd service can be configured with an automatic restart policy, allowing Chromium to relaunch immediately without manual intervention. To avoid session loss in such events, the application is intended to synchronize its state with the backend. Upon restart, it can retrieve and restore the previous session, eliminating the need for users to re-scan items.

6.5.2 Supermarket Layout Visualization

To visualize a supermarket layout, an easily modifiable approach is required - one that can be adapted for future deployment in a real supermarket environment. The solution must also remain lightweight and resizable to ensure crisp visual quality across different screen sizes.

An effective method is to use a matrix-based tile system that programmatically generates the entire layout. In this approach, each cell in a 2D matrix maps to a specific visual tile, represented by a distinct character. For instance, one character may denote walkable paths, while others represent aisles, checkout counters, or other key landmarks.

These visual tiles can be styled using CSS, enabling dynamic rendering of the entire layout without gaps, creating a smooth appearance. On top of this base layout, SVG elements can be layered to visualize icons and annotations, making it possible to mark product locations and display labels.

Figure 12. *Example of matrix-based tile rendering.*

This matrix-based approach allows for easy configuration changes by simply modifying the data structure without editing the rendering code.

6.6 Backend Architecture and Logic Considerations

6.6.1 API Gateway

The backend requires an API layer to serve as the communication interface between the frontend React application and the server-side logic, as it provides several critical advantages. First, the API decouples the frontend from backend implementation details, ensuring that changes to business logic or database structure do not require frontend modifications. This separation reduces development complexity and supports independent evolution of system components. Second, APIs enable support for multiple client types - mobile applications, web interfaces, and desktop software, allowing the same backend to serve various platforms without requiring separate implementations. If the system expands to include non-web clients, or even if the user interface is eventually replaced with a mobile application, the existing backend infrastructure can remain unchanged. Lastly, the API layer enforces security by preventing unauthorized access to sensitive data and validating all incoming requests.

To implement the backend, two primary options emerge when working with a React frontend - Node.js, which maintains JavaScript consistency across the full stack, and Python, which offers distinct advantages for this application domain. Python provides a rich ecosystem of libraries for optimization tasks and data processing necessary in this project as will be discussed in Section 6.6.5, along with a clear and expressive syntax that can reduce development effort. Moreover, since machine learning (ML) - for enabling personalized promotions, and computer vision (CV) - for detecting discrepancies between scanned and in-cart products, are commonly used in smart cart systems, Python's proven library landscape for both ML and CV makes it well-suited for future integration of such capabilities.

Therefore, FastAPI, a modern web framework for building APIs with Python, is selected for developing the API due to its performance characteristics, which are essential for handling concurrent requests from multiple shopping carts simultaneously. This is made possible by FastAPI's asynchronous capabilities, which allow the backend to efficiently manage high loads during peak usage without blocking operations, thereby maintaining responsive client devices. Additionally, the framework provides automatic API documentation generation, giving interactive visibility into all available endpoints and significantly simplifying development.

6.6.2 Data Collection and Management Strategy

While in a real deployment scenario the system would ideally integrate with the supermarket's existing database, the current project focuses on developing a functional prototype. Therefore, a custom database must be created from scratch, in order to demonstrate the concept.

The database in this project must include information about users and their preferences, purchase history, pre-loaded shopping lists, and detailed product data. The product information should include, among other attributes, barcode number, ingredients, detailed nutritional values, allergens, dietary suitability indicators, availability status and the physical location of each product within the supermarket. Collectively, this is the critical data to ensure the functionality of the system core capabilities.

For fault tolerance in the event of a GUI or hardware crash, each user's shopping session shall be periodically saved to the database. Recording every individual action, such as quantity adjustments, would be inefficient and unnecessarily resource-intensive.

Unlike traditional relational databases, NoSQL databases offer greater flexibility, making them well-suited for fast prototyping and iterative development. Both MongoDB and Firebase provide generous free tiers and simple setup processes, making them attractive options for this project. However, MongoDB is chosen due to its support for organizing data into collections, which function similarly to relational tables and allow for more structured data modeling. In contrast, Firebase stores data in a deeply nested JSON structure, which can complicate complex queries and relationships between entities - a limitation that makes it less suitable for the needs of this system.

Since no publicly available dataset exists for Israeli supermarket products, and most commercial websites restrict automated data collection via web scraping, product information for this project will be gathered manually. As this is a prototype, the scope will be intentionally limited to non-weighted food products only. Furthermore, to keep the development effort focused on system functionality rather than data collection, the database will include only a few dozen representative products. This constrained dataset is sufficient for demonstration.

Similarly, product location data and the overall supermarket layout are essential for enabling product location guidance and route optimization. To obtain this information, a real mid-sized supermarket will be observed and documented in order to create a realistic supermarket map with accurate product placements.

6.6.3 Recommendation Modules Design

While AI can be perceived by the user as the entity responsible for proactively recommending alternative products when scanned items conflict with their restrictions, this functionality does not require complex language understanding or AI-based reasoning. Invoking LLM in such cases would consume substantial computational resources for a problem that can be efficiently addressed using a lightweight, rule-based algorithm.

When a conflict is detected, the system searches the database for alternative products within the same category that do not violate any of the user's constraints. This ensures that the alternatives are both relevant and compliant. Once identified, these suggestions are displayed to the user through the digital companion panel, accompanied by a pre-generated audio response to preserve the illusion of real-time speech by the companion.

In contrast, the reactive mode, which operates upon user inquiry - does invoke the LLM. User requests are phrased in natural language and often require contextual understanding and reasoning to compare attributes such as nutritional values between the scanned product and others in the same category, as well as to deliver meaningful explanations that enhance user trust and decision confidence. Therefore, in order to generate appropriate alternative suggestions, the LLM utilizes function calling to retrieve relevant products data from the database, analyze them, and produce informed, conversational responses.

As for the frequently-bought products suggestions module, which is triggered at checkout as a pop-out, the algorithm requires thresholds for both number of past purchases to consider and the minimum frequency with which an item must appear across those sessions. These thresholds help strike a balance between usefulness and relevance, ensuring that only frequently purchased essentials are suggested, while avoiding items that are either already in the cart or were rarely bought in the past.

This introduces several design considerations. For instance, looking too far back across many past shopping sessions may result in irrelevant suggestions, as users' preferences may have changed over time or products may be seasonal. On the other hand, considering only the most recent purchase may fail to capture consistent habits. Therefore, the thresholds must be carefully calibrated to capture meaningful patterns while filtering out noise.

6.6.4 Product Availability and Mapping Logic

As inquiries are expressed in natural language, user requests for product locations can range from specific branded products, such as 'Tnuva 3% Milk', to broader product categories, such as 'Milk'. For the LLM to support this, each specific product in the database must be linked to a corresponding category, with location data stored only at the category level as a cell within the 2D matrix described in Section 6.5.2. These cells are then used to mark the relevant area on the supermarket map layout.

The LLM must also be stock-aware. When a location or availability inquiry refers to a broader product category, the LLM should evaluate the stock status of all specific products within that category. If none are available, it can conclude that the category is effectively out of stock, since availability is tracked at the individual product level.

6.6.5 Route Optimization

Implementing an efficient solution for the TSP from scratch presents significant technical challenges that go far beyond the core algorithmic logic. It demands a deep understanding of optimization techniques and considerable time investment for development, testing, and fine-tuning - often outweighing the benefits, especially when robust solutions already exist.

One such solution is Google's Operations Research Tools (OR-Tools), a Python library that was developed, optimized and tested over years. It provides advanced algorithms for solving problems such as the Vehicle Routing Problem (VRP), which is a generalization of TSP involving multiple "salesmen" (or vehicles). By leveraging OR-tools, we can focus on the application-specific logic.

The following demonstration illustrates how OR-Tools can be applied to compute a near-optimal yet efficient pickup order for a predefined shopping list. For this purpose, a simplified supermarket layout is used (see Figure 13). In this layout, white cells represent walkable paths, while gray cells represent walls or shelves. Green circles labeled ‘A’, ‘B’, and ‘C’ denote the locations of products to be picked up, while the red circle labeled ‘S’ marks the starting point at the supermarket entrance.

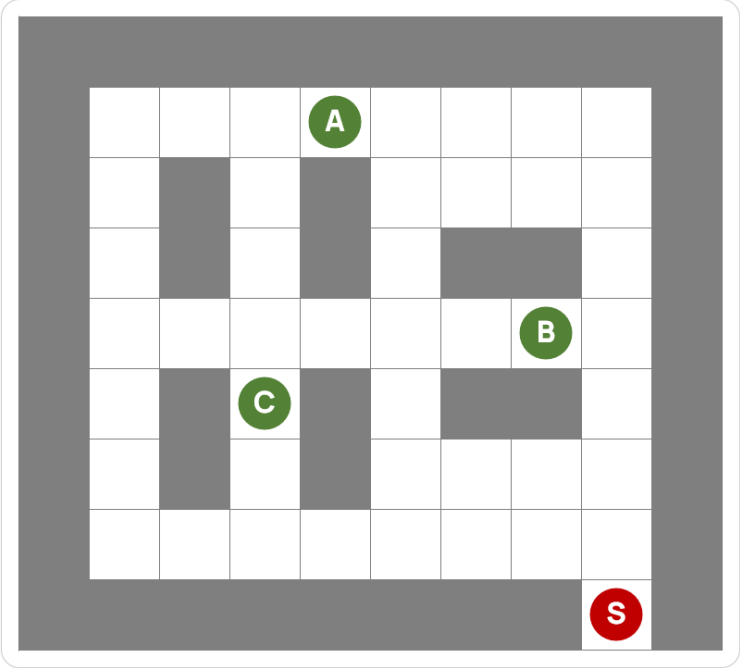

Figure 13. *Simplified supermarket layout.*

To apply the routing solver provided by OR-tools, the following parameters are required:

1. A distance matrix, specifying the minimum walkable cells – representing the shortest walkable path between the nodes (the entrance cell and the pickup location cells).
2. The number of vehicles, set to 1 as this is a TSP scenario (a single customer traversing the route).
3. The start and end node for the customer. In this demonstration, both are assigned to the same location labeled ‘S’, which corresponds to the supermarket entrance.

For example, the shortest walkable path between nodes A and C, accounting for walls and shelves, consists of 6 walkable cells. The path is illustrated below.

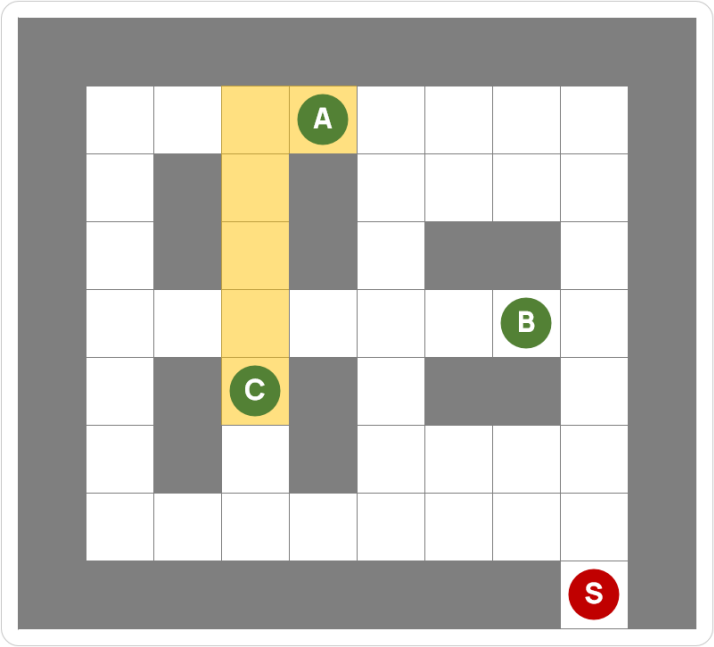

Figure 14. *Visualization of the shortest walkable path between product locations 'A' and 'C'.*

With these parameters, the routing solver first uses a method, which must be defined, to construct an initial route between the nodes. A common method, called "PATH_CHEAPEST_ARC" [55], which corresponds to the nearest neighbor algorithm mentioned in Section 2.2.4. Then, the solver uses this initial route as a valid starting point and applies a metaheuristic method called Guided Local Search to improve it, as more efficient routes may exist, producing a near-optimal yet efficient solution.

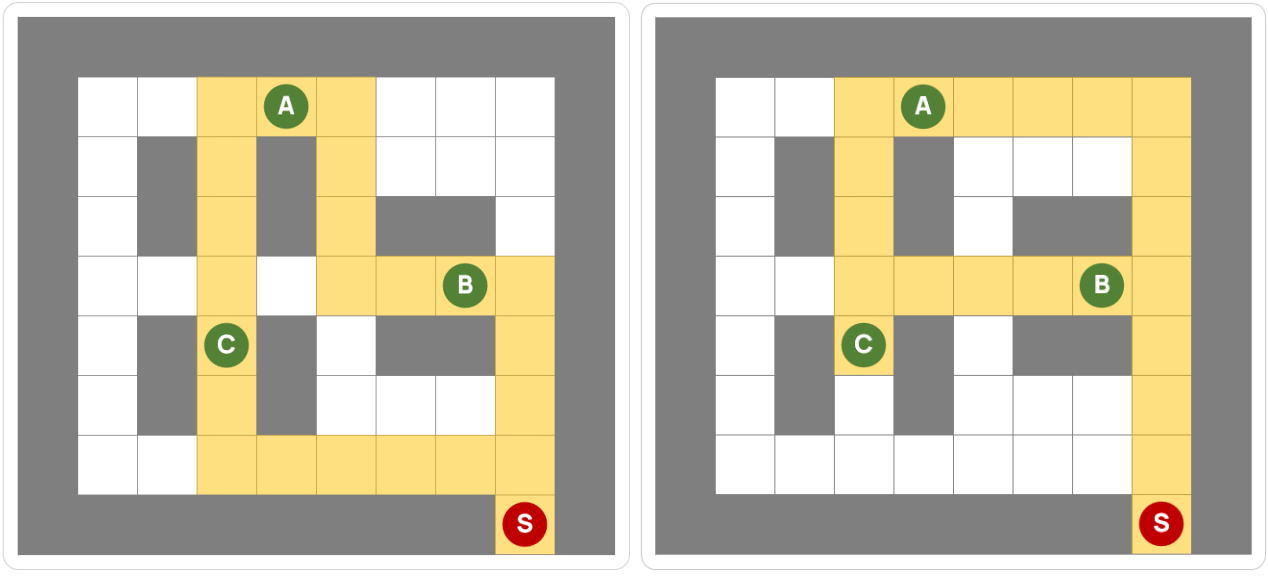

Figure 15. *Greedy initial route shown on the right (27 steps) and optimized route (25 steps).*

Following the explanations, the implementation in code is straightforward, as shown below.
The result produced by the solver corresponds to the optimized route shown in Figure 15.

```
# 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]
```

Figure 16. *Routing solver implementation and resulting output.*

In real deployment, the system must operate over a large matrix representing the shortest walking distances between product categories. At runtime, only the relevant subset of nodes - based on the user's shopping list, must be extracted. To ensure this filtering process is efficient, especially as the full matrix scales, a performant data structure such as a NumPy array shall be used due to its speed and memory efficiency. The distance matrix itself can be constructed manually for smaller layouts, but for larger supermarket maps, it is more effective to compute the shortest paths between all node pairs using algorithms such as A* or by performing breadth-first-search (BFS) from each node. It is also important to note that the system does not provide the full path itself. However, the pickup order still offers a meaningful understanding of the route to be taken.

6.7 Necessity of Edge Services on the Raspberry Pi

Most USB barcode scanners operate as Human Interface Devices (HID), effectively emulating keyboard input. When a barcode is scanned, the scanner transmits the corresponding numeric digits as keystrokes, which are automatically directed to the currently focused interface component - in this case, the shopping cart GUI. To ensure reliable data capture, it is necessary to isolate scanner input from the GUI, as this default behavior poses a risk of unintended events.

To address this, a dedicated service is deployed on the Raspberry Pi to manage barcode input. This service takes exclusive control over the scanner by “grabbing” the device [56], thereby ensuring that only it receives input events. This is achieved using the evdev library, a low-level Python interface for Linux input event devices that enables direct communication with hardware peripherals. Upon scanning, the service captures the barcode and forwards the decoded value to the GUI. The GUI then communicates with the backend to retrieve the corresponding product information and display it to the user.

Additionally, a second local service is deployed to manage voice interactions with VAPI, handling both microphone input and speaker output. While it is technically possible to enable basic voice functionality through the browser, assigning this task to a separate local service will enable custom sample rate tuning and noise reduction, which require direct access to the audio hardware - something that the web-based GUI does not support. Although these advanced features are outside the scope of the current prototype, this architecture provides a solid foundation for adding them in future development stages.

6.8 Methodology and Development Process

To manage the development process in this project, the Agile methodology is essential, as it allows for changing requirements which are likely to arise in this kind of project, and enables early detection and correction of errors. The process is divided into the following key phases, based on the discussion in this chapter. While the phases are presented in logical order, it is important to note that some tasks may overlap or carried out in parallel, in accordance with Agile principles.

Table 6. *Development key phases*

No.	Phase	Tasks	Order Rationale
1	Data Collection and Setup	Gather product data, document supermarket layout, create database	Provides the core data required by backend features
2	Initial Raspberry Pi Setup	Install Raspberry Pi OS and connect peripherals	Prepares the hardware and verifies that peripherals work
3	Voice Assistant Development	Implement the Python service that interacts with VAPI, write the behavior prompt for the VAPI agent and deploy backend logic required for function calls	Enables early testing of the voice interaction pipeline
4	Barcode Scanner Service Development	Implement the Python service that captures barcode scanner input using the evdev library	Ensures scanner input is available for integration and UI testing
5	UI Development and Further Logic Implementation	Build React components, complete and deploy additional backend logic required (including product location, list creation and routing, authentication), integrate the UI with local Raspberry Pi services, and deploy the UI to a remote web platform	Allows testing the full experience on the hardware and perform adaptations
6	Raspberry Pi Boot Configuration	Configure Chromium to run in kiosk mode, and set up the systemd service for automatic startup and recovery	Final step that prepares the system for autonomous operation

7 Work Artifacts

7.1 Product Requirements

Table 7. *Functional requirements.*

No.	Requirement
1	The system shall support OTP verification for user authentication
2	The system shall allow users to create a shopping list
3	The system shall display supermarket map with near-optimized pickup order
4	The system shall allow products to be added and removed from the cart list
5	The system shall support AI-driven interactions for user inquiries
6	The system shall provide personalized warnings along with suitable alternative product suggestions
7	The system shall enable to replace product on the cart list with suggested alternative
8	The system shall display total price
9	The system shall enable checkout
10	The system shall suggest products that may have been forgotten by the user

Table 8. *Non-functional requirements.*

No.	Requirement	Type
1.1	The OTP is delivered via SMS	Security
1.2	OTP must expire within a defined session timeout	Security
1.3	New users are automatically created without registration steps	Usability
2.1	List creation is performed on an external device such as smartphone, tablet or personal computer	Interoperability
3.1	The pickup order is displayed when the user loads a pre-created shopping list	Usability
3.2	Products are shown in numbered order, with matching numbers on the map	Usability
3.3	Products located in the same place are assigned the same number	Usability
4.1	Products are added to the cart via barcode scanning	Usability
4.2	Quantity changes are done manually through the cart list panel	Usability
5.1	The interactions are voice-based	Usability
5.2	Follow-up questions may be asked by the voice assistant to: gather missing details, better understand user dietary needs	Adaptability
5.3	Users may refer to the last scanned product implicitly when asking a question	Usability
5.4	Voice-based inquiries are limited to product-related topics: information, location and alternatives suggestions	Usability
5.5	Inquiry about specific product that is out of stock results notification about unavailability	Usability
5.6	Inquiry responses shall rely on verified product data to ensure trustworthy information	Reliability
6.1	Personalized warnings are shown when a product conflicts with the user's allergens or dietary restrictions	Safety
8.1	Total price reflects scanned products only	Reliability
9.1	Checkout is enabled when all listed products on the cart list panel are scanned	Usability
10.1	Forgotten product suggestions are displayed upon initiating checkout	Usability
10.2	Forgotten product suggestions are based on the user's previous purchases	Adaptability
11	All inquiries and system responses including alerts and maps shall be shown within the chat panel	Usability
12	The system database shall include: product names, barcodes, nutritional information, location and availability; user phone number, dietary needs and shopping history	Data Integrity
13	The smart cart interface shall automatically recover from unexpected crash with all loaded items	Fault Tolerance
14	The system shall be centrally updated	Maintainability
15	The system interfaces shall be compatible and responsive across all devices	Compatibility
16	Concurrent usage by multiple smart carts shall not cause system shutdown	Stability
17	System responses shall be provided within reasonable time (< 3 seconds)	Performance

Note. The hierarchical numbering of non-functional requirements corresponds to their related functional requirements.

7.2 UML Diagrams

7.2.1 Use Case Diagram

The use case diagram illustrates how customers and external systems such as VAPI and Twilio interact with the system to perform specific tasks.

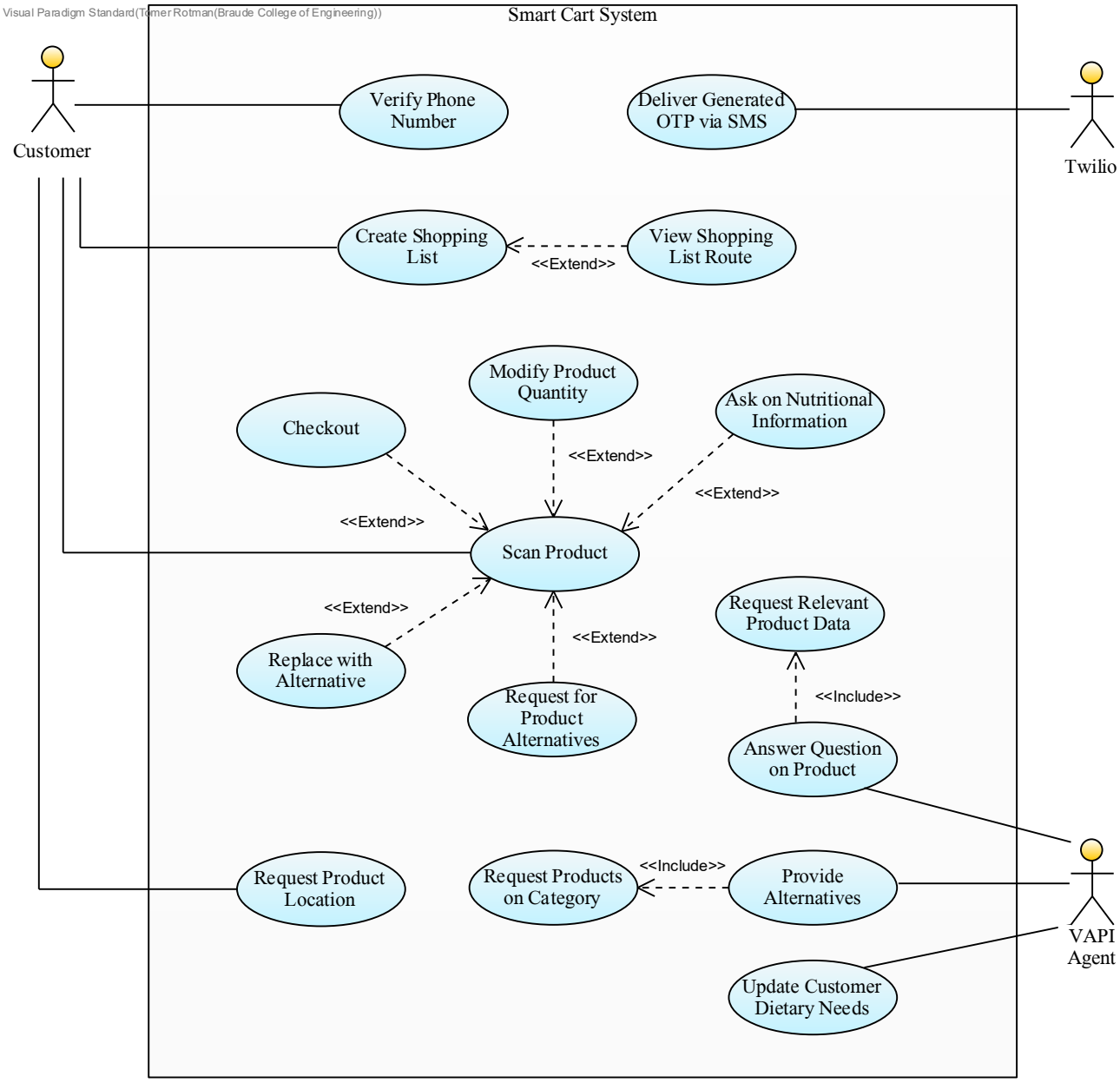

Figure 17. *The smart cart system Use Case Diagram.*

7.2.2 Activity Diagram

The activity diagram illustrates the dynamic flow of actions that occur during a typical shopping session using the smart cart system.

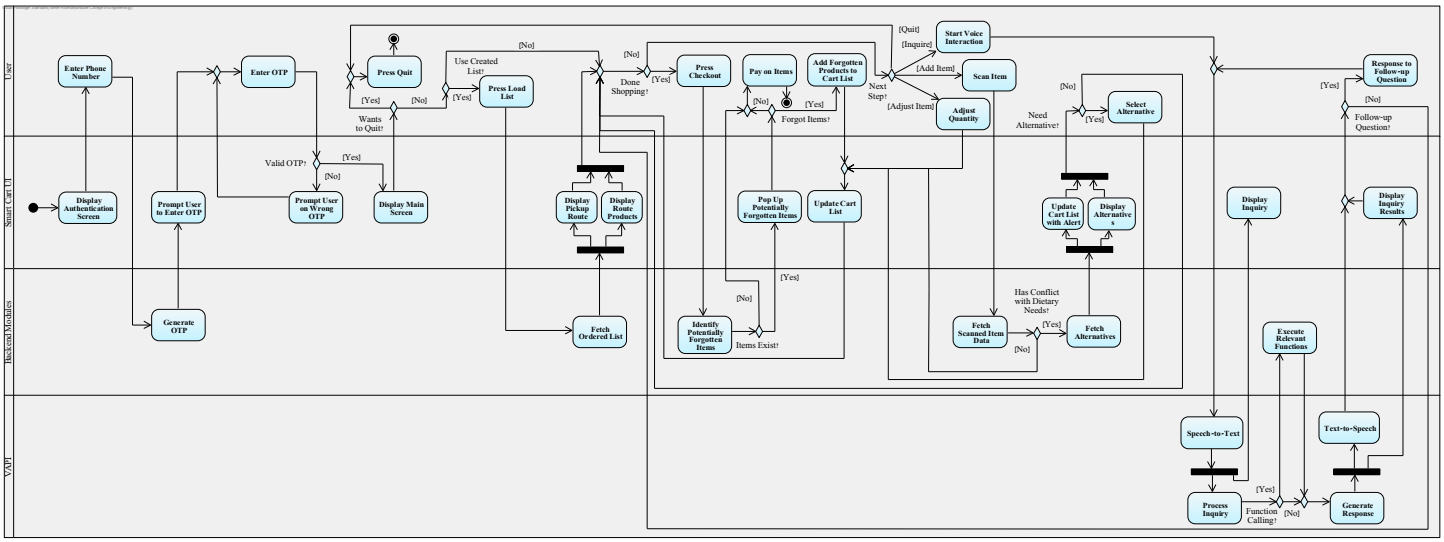

Figure 18. *The smart cart system Activity Diagram.*

7.3 Software Architecture and Deployment

The engineering process resulted in the software architecture shown below.

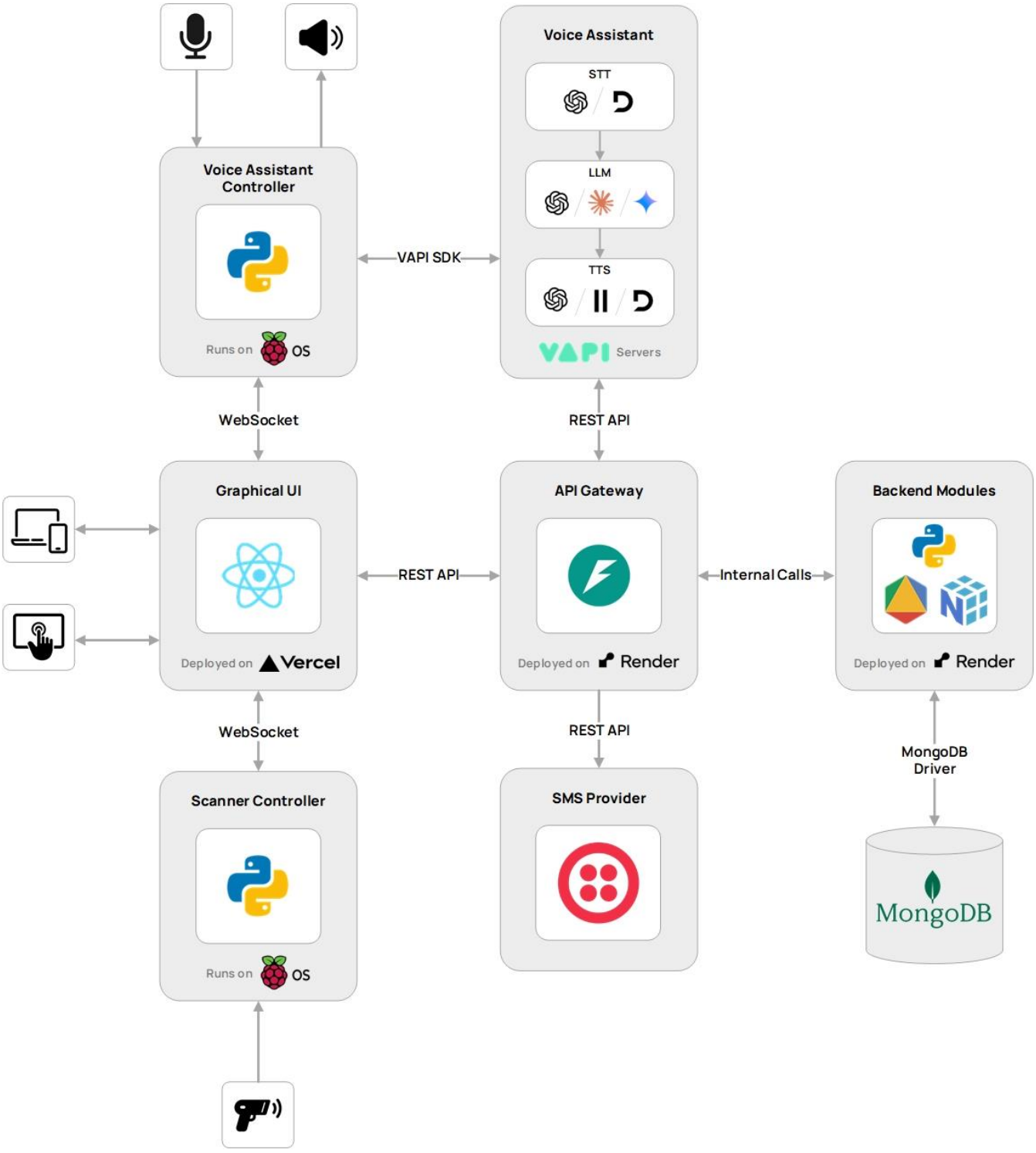

Figure 19. *The smart cart system software architecture.*

As illustrated in the architecture diagram, there are two main interaction points with the system: the user's personal device and the smart cart. Both interaction points operate within a shared full-stack environment - running the same React-based web application and interacting with a centralized FastAPI backend. While they share the same core infrastructure, each interaction point serves a different purpose and provides access to different tools and functionalities.

All client-to-server communication is conducted through RESTful HTTP requests. These requests trigger backend functions for data processing or logic execution, with structured responses returned in JSON format.

From the user's personal device, interactions are initiated primarily through API calls - such as when uploading a shopping list. Upon receiving this request, the backend filters the product distance matrix using NumPy and forwards the relevant subset to the OR-Tools solver. The generated pickup route is then stored in the MongoDB database, making it accessible to the smart cart interface.

The smart cart communicates with local controllers using bidirectional WebSocket connections.

Barcode scan events are transmitted from the Scanner Controller to the UI, which then performs a RESTful request to fetch matching product data from the backend. This ensures the cart is updated in real time without polling overhead.

Voice interactions on the smart cart are initiated by the UI through a WebSocket message sent to the Voice Assistant Controller, which communicates with VAPI via the VAPI SDK. The message includes context, such as the last scanned product's barcode. Once the voice interaction completes, VAPI returns a structured response over the same WebSocket channel, which is then displayed in the UI. VAPI also interacts with the backend by issuing RESTful API calls to retrieve information or trigger logic as needed.

When the backend generates an OTP, it sends it to Twilio via a RESTful API.

As the system requires centralized maintainability, the web application is deployed on Vercel, which is optimized for React-based applications, while the backend is deployed on Render, which supports FastAPI-based services. Both platforms offer free tiers.

7.4 User Interface

This section presents illustrations of both the envisioned list-creation interface and the smart cart interface.

The list-creation interface can run either on smartphones, tablets or personal computers. On the smartphone, the interface consists of two tabs: one for browsing items and adding them to the list, and one for viewing the list and adjusting quantities. On a tablet or personal computer, the same functionality is presented using a split-screen layout instead of tabs. The user can monitor the running total, and once finalized, upload it to the smart cart system using the “Upload List” button.

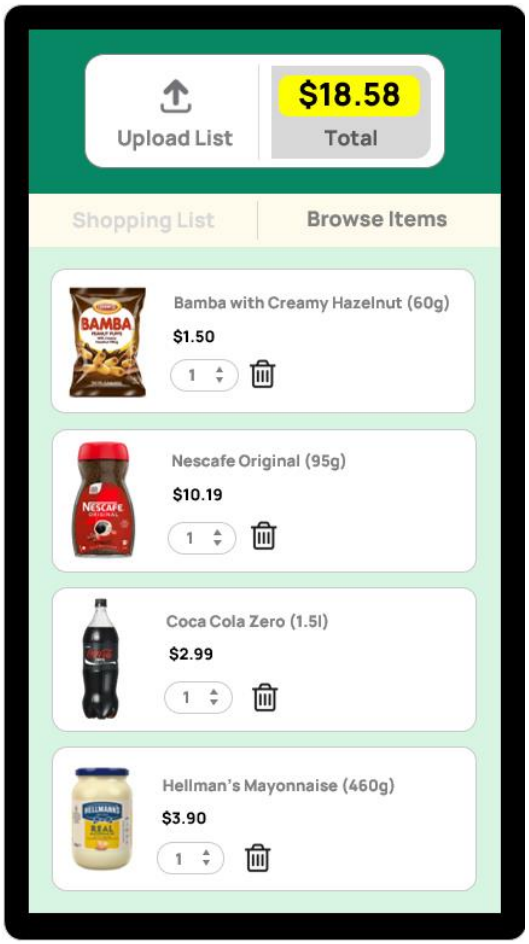

Figure 20. *The list-creation interface on a smartphone, displaying the selected items.*

The smart cart interface is composed of two scrollable panes, one displaying the interaction history with the digital companion, and the other listing the items to be collected or already added to the cart. A top navigation bar provides access to functional buttons and displays key information.

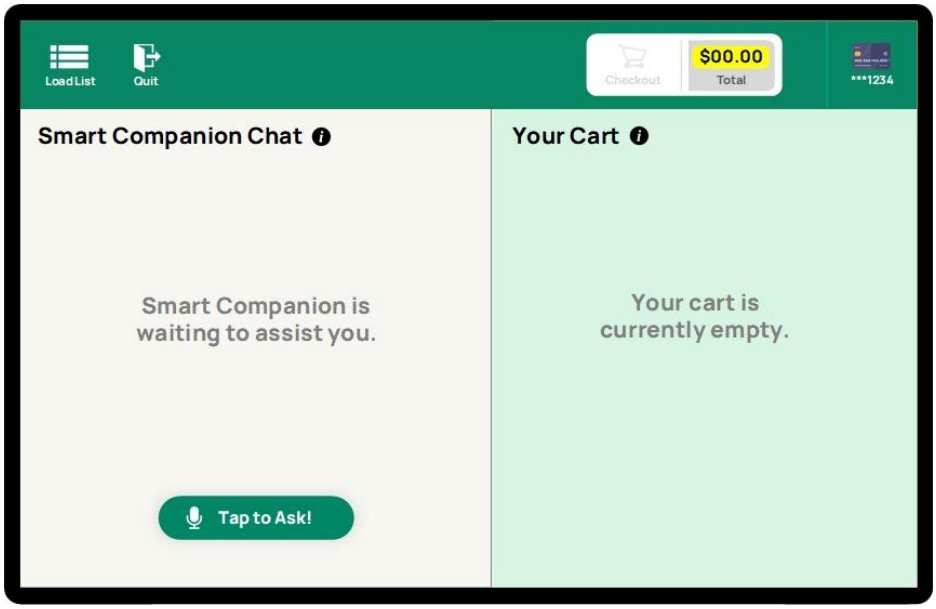

Figure 21. *The smart cart interface at the beginning of the shopping process.*

In case the user has prepared a shopping list in advance through the system's dedicated list-creation interface, it can be loaded into the system to display the supermarket map with a recommended item pickup order. The user must scan all listed items before being allowed to proceed to checkout.

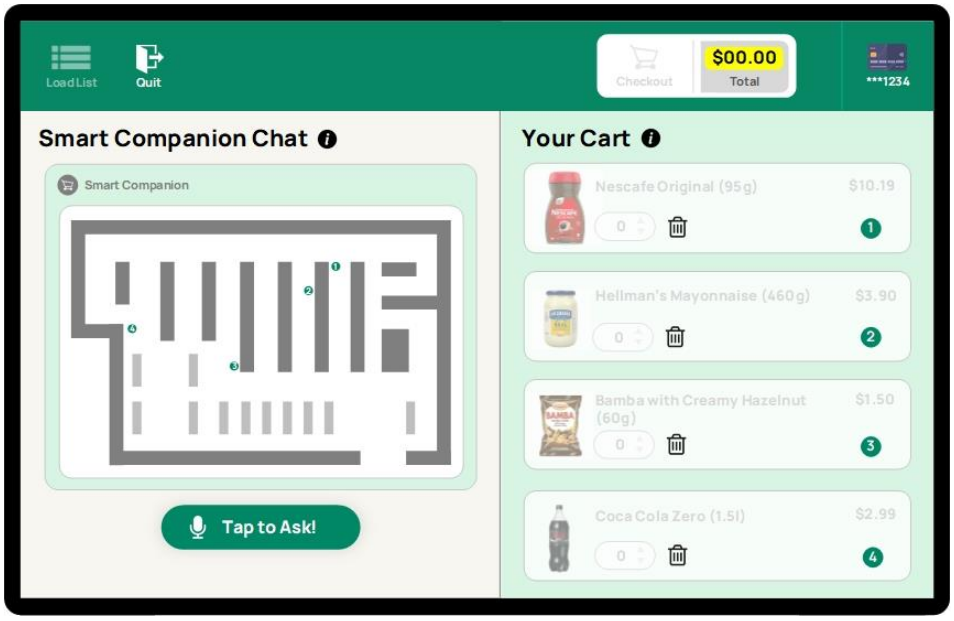

Figure 22. *Displaying a pre-created list with a near-optimal pickup order.*

Using the “Tap to Ask!” button, the user can request a product alternative via voice. In response, the digital companion provides a rationale for the suggested substitutes and enables the user to replace the originally scanned item.

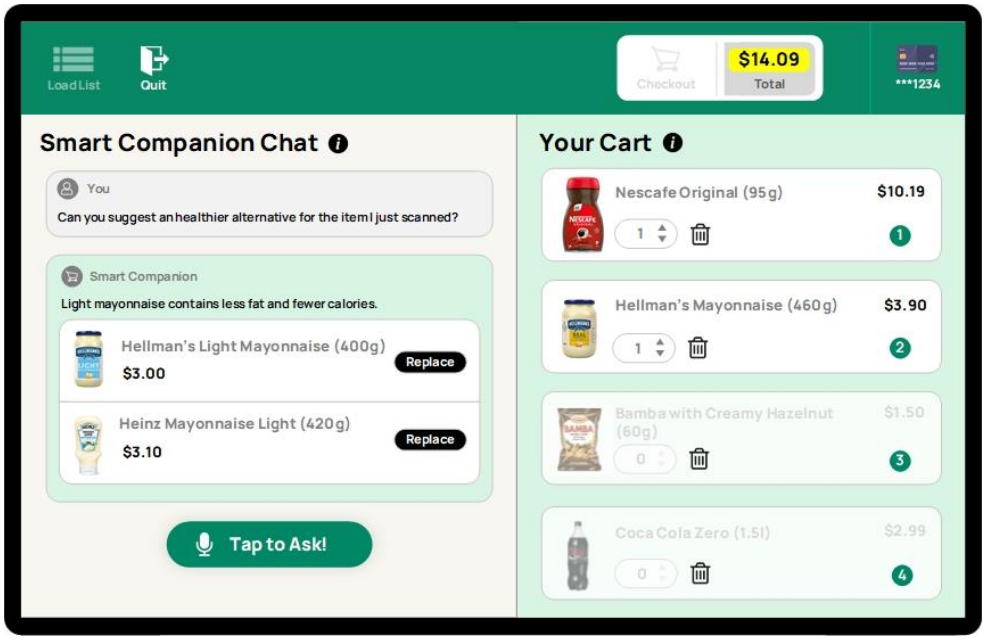

Figure 23. *The reactive alternative product suggestions module in operation following a user inquiry.*

When an item that conflicts with the user's preferences is scanned, the interface highlights the conflict and displays an alert through the digital companion interaction panel, accompanied by suitable replacement options.

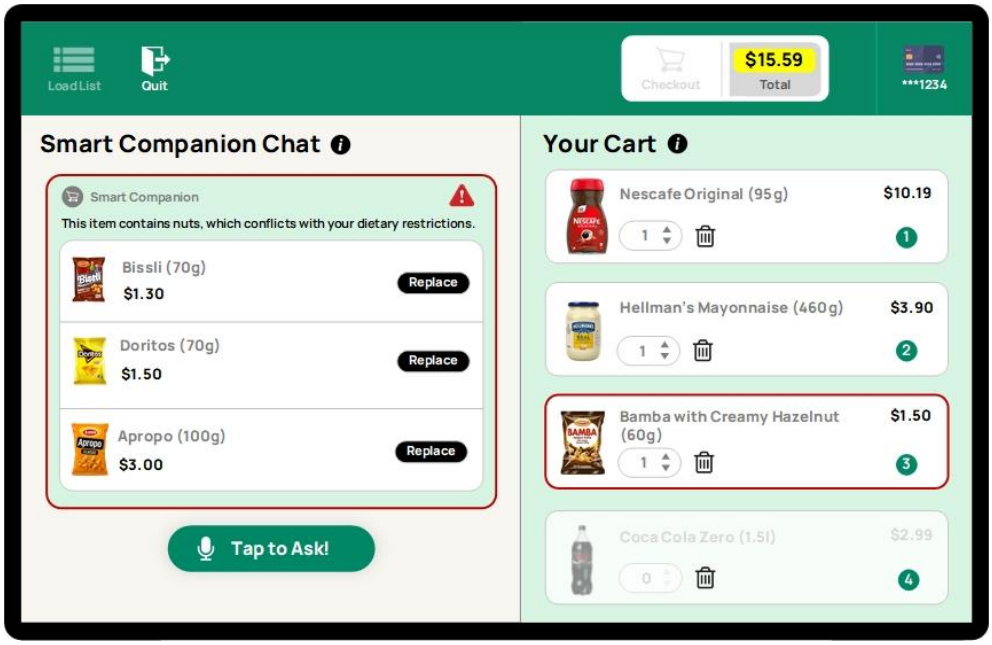

Figure 24. *Personalized warning displayed alongside proactive alternative suggestions.*

When proceeding to checkout, a pop-out appears displaying frequently purchased items from previous shopping sessions. If at least one item is selected, the checkout button changes to a “Return to Session” button, and the selected items are added to the list for scanning. If no additional items are needed, clicking “No, Thanks!” will complete the process.

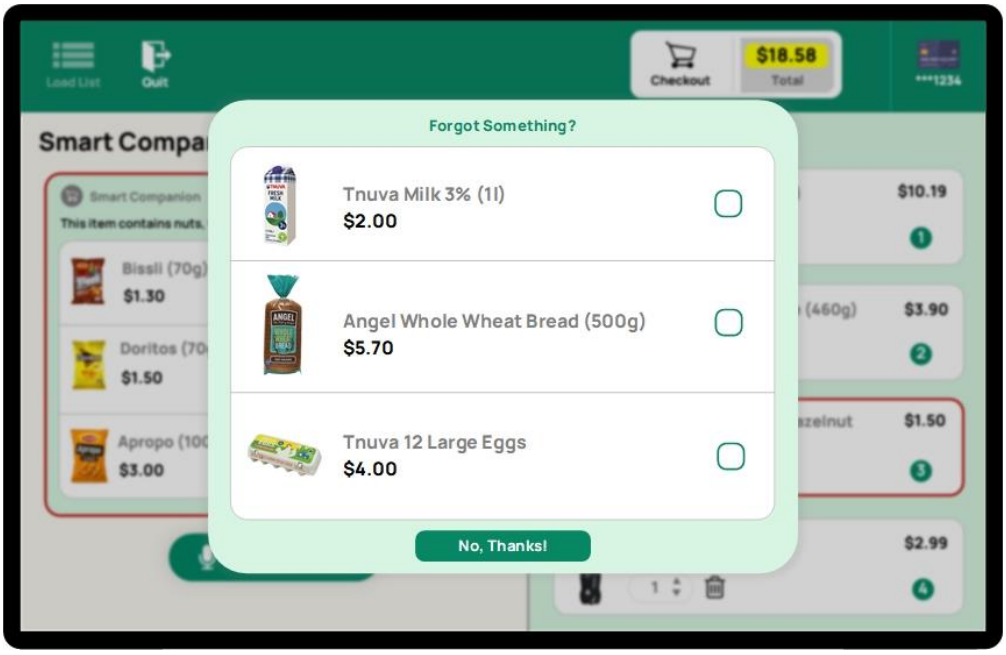

Figure 25. *Frequently-bought products suggestions module in operation during the checkout process.*

8 Evaluation and Verification

8.1 Evaluation

Whether the system meets the success criteria defined in Section 5.2 will be determined through an evaluation process.

The impact of optimization strategies on cost and latency will be evaluated using VAPI’s built-in analytics tools. For each optimization iteration, the same scripted interactions will be used to ensure consistency. This will allow direct comparison of total interaction cost, as well as response times, across different optimization approaches.

System usability and assistant response accuracy will be assessed through testing with volunteer participants upon completion of development. Each participant will complete a SUS questionnaire, and their conversations with the assistant will be documented and reviewed to determine whether the assistant’s responses correctly matched the intended user inquiries.

System stability will be evaluated manually by running multiple devices in parallel, each performing scripted voice interactions to replicate realistic concurrent usage.

Insights from the evaluation will inform final adjustments and confirm the system’s viability under real-world conditions.

8.2 Verification

The verification process ensures that the prototype functions correctly according to the requirements, including its underlying system components and integration dependencies. In line with the Agile development methodology, tests are conducted iteratively alongside implementation. Accordingly, it involves a combination of component-level validation and integration testing to confirm that each component performs its assigned task and that the overall system operates in a coordinated and consistent manner.

Table 9. *Verification test plan*

No.	Components	Test	Method	Expected Results
1	Raspberry Pi	Network Connectivity Check	Ping known host after OS installation	Successful ping response, indicating network access
2	Hardware Peripherals	Peripherals Sanity Check	Manually interact with each peripheral after Raspberry Pi OS installation	- Touchscreen responsive - Scanner outputs digits - Audio plays - Microphone records
3	Voice Assistant Controller, VAPI, Backend, Database	Voice Inquiry Test	Ask a question that results in a database query	- Response received within 3 seconds - Response speech is audible - Correct function was invoked - Accurate information provided
4	Voice Assistant Controller, VAPI, Backend, Database	Dietary Profiling Test	Ask a product-related question that implies a dietary concern	- Information retrieved, followed by confirmation of potential dietary restriction - User preference is stored in database
5	Voice Assistant Controller, VAPI, Backend, Database	Clarification Request Test	Ask incomplete question, simulating noise or unclear phrasing	- Assistant asks a clarifying question to gather missing information - Response is provided only after clarification
6	Voice Assistant Controller, VAPI, Backend, Database	Alternative Request Test	Ask for an alternative using an injected test barcode	- Assistant provides alternatives list - Assistant provides good explanation
7	Voice Assistant Controller, VAPI, Backend, Database	Location Request Test	Ask for product locations using both specific product names and general categories	Assistant provides the correct temporary cells assigned for testing, or notifies the user about product unavailability
8	Voice Assistant Controller, VAPI	Out-of-Scope Inquiry Test	Ask a question unrelated to the assistant scope	Assistant politely declines the user
9	Scanner Controller	Barcode Transmission Test	Scan a barcode of an item in the database	Controller transfers the barcode to a test webpage
10	UI	Responsiveness Test	Test various screen sizes in kiosk mode	Components are resized or rearranged without overlapping, cut-off, or misalignment

Table 9 *(Continued)*

No.	Components	Test	Method	Expected Results
11	UI, Scanner Controller, Backend, Database	Cart Update and Dietary Warning Test	Scan product barcodes and observe cart updates on the UI	- Products appears in cart list within 1 second - Product details are displayed accurately - Warning and alternatives are shown in case of conflict - Total cost changes per scan
12	UI	Cart Quantity Adjustment Test	Scan product and adjust quantity	- Product quantity adjusted accurately - Reducing the quantity below 1 removes the product from the cart - Pressing the “Delete” button removes the product from the cart - Total cost changes per adjustment
13	UI	Product Replacement Test	Replace an item in the cart with a suggested alternative and observe cart behavior, cost adjustment, and checkout availability	– The alternative product is displayed in the cart list instead of the original product - Checkout remains disabled until the alternative product is scanned or removed - Original product cost is reduced from the total cost
14	UI, Voice Assistant Controller, Backend, Database	Chat Panel Update Test	Ask questions and observe matching updates on the chat panel	After each assistant interaction, the chat panel is updated with the full message and any supporting components, such as an alternatives list or product location on the map, if applicable
15	UI, Backend, Database	List Creation Test	Create product list with more than 50 products and update to the system	- Solving the route takes less than 1 second - Pickup order is saved in the database - Products located in the same area are assigned the same pickup number
16	UI, Backend, Database	Post-Load Cart Interaction Test	Load a pre-created product list and verify cart display, map visualization, product scanning behavior, and checkout availability	- All products are displayed in the cart list within 3 seconds - Pickup order is shown on the map with matching numbers on both products and map locations - Scanning a product increments its quantity up to the predefined amount from the list and total cost changes - Checkout is enabled only when all products in the cart list are scanned

Table 9 *(Continued)*

No.	Components	Test	Method	Expected Results
17	UI, Backend, Database	Frequently-Bought Products Suggestions Module Test	Attempt to proceed to checkout without scanning frequently purchased products defined in a mock purchase history	- Frequently purchased products that were not added to the cart are shown in a pop-up window - Selecting at least one suggested product replaces the “No, Thanks!” button with “Continue Shopping”
18	UI, Backend, Database	Accept Suggestions and Continue Shopping Test	Following the pop-up of the frequently-bought products module, press the “Continue Shopping” button, verify behavior and checkout availability	- User is returned to the cart screen - Selected items are added to the cart list - Checkout remains disabled until the newly added items are scanned
19	UI, Backend, Database	Skip Recommended Products Test	Following the pop-up of the frequently-bought products module, pressing the “No, Thanks!”	- Purchase session is saved in the database - User is redirect to the authentication page
20	UI, Backend, Database	Authentication Test	Insert a phone number and attempt to authenticate	- OTP SMS is sent to the specified phone number - On successful verification, the user is redirected to the main page - If the user does not exist, a new entry is created in the database
21	Service systemd, UI, Voice Assistant Controller, Scanner Controller, Backend, Database	Crash Recovery and Session Restore Test	Close the UI and controllers and observe behavior	- The systemd service automatically relaunches the UI and controllers without manual intervention - After re-authentication, most of the previous session is restored

9 References

[1] Wang, S., Ye, Y., Ning, B., Cheah, J. H., & Lim, X. J. (2022). Why do some consumers still prefer in-store shopping? An exploration of online shopping cart abandonment behavior. *Frontiers in Psychology*, *12*, 829696. https://doi.org/10.3389/fpsyg.2021.829696

[2] Van Riel, A. C., Semeijn, J., Ribbink, D., & Bomert-Peters, Y. (2012). Waiting for service at the checkout: Negative emotional responses, store image and overall satisfaction. *Journal of service management*, *23*(2), 144-169. https://doi.org/10.1108/09564231211226097

[3] Bloemer, J., & De Ruyter, K. (1998). On the relationship between store image, store satisfaction and store loyalty. *European Journal of marketing*, *32*(5/6), 499-513. https://doi.org/10.1108/03090569810216118

[4] Schultz, C. D., & Zacheus, P. (2025). Smart shopping carts in food retailing: Innovative technology and shopping experience in stationary retail. *Journal of Consumer Behaviour*, *24*(1), 436-454. https://doi.org/10.1002/cb.2426

[5] Bohns, V. K., & Flynn, F. J. (2010). "Why didn't you just ask?" Underestimating the discomfort of help-seeking. *Journal of Experimental social psychology*, *46*(2), 402-409. https://doi.org/10.1016/j.jesp.2009.12.015

[6] Fan, H., & Poole, M. S. (2006). What is personalization? Perspectives on the design and implementation of personalization in information systems. *Journal of Organizational Computing and Electronic Commerce*, *16*(3-4), 179-202. https://doi.org/10.1080/10919392.2006.9681199

[7] Chen, J., Liu, Z., Huang, X., Wu, C., Liu, Q., Jiang, G., ... & Chen, E. (2024). When large language models meet personalization: Perspectives of challenges and opportunities. *World Wide Web*, *27*(4), 42. https://doi.org/10.1007/s11280-024-01276-1

[8] Food Allergy Research & Education. (n.d.). *Facts and statistics.* https://www.foodallergy.org/resources/facts-and-statistics

[9] Warren, C. M., Jiang, J., & Gupta, R. S. (2020). Epidemiology and burden of food allergy. *Current allergy and asthma reports*, *20*(2), 6. https://doi.org/10.1007/s11882-020-0898-7

[10] Neuhouser, M. L. (2019). The importance of healthy dietary patterns in chronic disease prevention. *Nutrition research*, *70*, 3-6. https://doi.org/10.1016/j.nutres.2018.06.002

[11] Leidy, H. J., Clifton, P. M., Astrup, A., Wycherley, T. P., Westerterp-Plantenga, M. S., Luscombe-Marsh, N. D., ... & Mattes, R. D. (2015). The role of protein in weight loss and maintenance. *The American journal of clinical nutrition*, *101*(6), 1320S-1329S.. https://doi.org/10.3945/ajcn.114.084038

[12] World Health Organization Regional Office for Europe. (2021, November 4). *Plant-based diets and their impact on health, sustainability and the environment: A review of the evidence* (WHO Reference No. WHO/EURO:2021-4007-43766-61591). Copenhagen: WHO Regional Office for Europe. https://www.who.int/europe/publications/i/item/WHO-EURO-2021-4007-43766-61591

[13] Global Food Research Program at UNC-Chapel Hill. (2025, January). *Front-of-package labeling fact sheet* (Updated January 2025). Retrieved from https://www.globalfoodresearchprogram.org/wp-content/uploads/2025/01/Factsheet_FOPL_Jan-2025.pdf

[14] Lacy-Nichols, J., Hattersley, L., & Scrinis, G. (2021). Nutritional marketing of plant-based meat-analogue products: an exploratory study of front-of-pack and website claims in the USA. *Public health nutrition*, *24*(14), 4430-4441. https://doi.org/10.1017/S1368980021002792

[15] Deakin, T. A. (2011). Consumers find food labels confusing and too small to read. *Practical Diabetes International*, *28*(6), 261-264c. https://doi.org/10.1002/pdi.1611

[16] Sansweet, S., Jindal, R., & Gupta, R. (2024). Food allergy issues among consumers: a comprehensive review. *Frontiers in Nutrition*, *11*, 1380056. https://doi.org/10.3389/fnut.2024.1380056

[17] Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2011). The traveling salesman problem: a computational study. In *The Traveling Salesman Problem.* Princeton university press. https://doi.org/10.1515/9781400841103

[18] Davendra, D. (Ed.). (2010). *Traveling salesman problem: Theory and applications.* BoD–Books on Demand. https://doi.org/10.5772/547

[19] Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., ... & Mian, A. (2023). A comprehensive overview of large language models. *ACM Transactions on Intelligent Systems and Technology*. https://doi.org/10.48550/arXiv.2307.06435

[20] Wu, J., Gan, W., Chen, Z., Wan, S., & Yu, P. S. (2023, December). Multimodal large language models: A survey. In *2023 IEEE International Conference on Big Data (BigData)* (pp. 2247-2256). IEEE. https://doi.org/10.48550/arXiv.2311.13165

[21] Li, J., Xu, J., Huang, S., Chen, Y., Li, W., Liu, J., ... & Dai, G. (2024). Large language model inference acceleration: A comprehensive hardware perspective. *arXiv preprint arXiv:2410.04466.* https://doi.org/10.48550/arXiv.2410.04466

[22] Samsi, S., Zhao, D., McDonald, J., Li, B., Michaleas, A., Jones, M., ... & Gadepally, V. (2023, September). From words to watts: Benchmarking the energy costs of large language model inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)* (pp. 1-9). IEEE. https://doi.org/10.48550/arXiv.2310.03003

[23] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30.* https://doi.org/10.48550/arXiv.1706.03762

[24] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

[25] Marvin, G., Hellen, N., Jjingo, D., & Nakatumba-Nabende, J. (2023, June). Prompt engineering in large language models. In *International conference on data intelligence and cognitive informatics* (pp. 387-402). Singapore: Springer Nature Singapore. https://doi.org/10.1007/978-981-99-7962-2_30

[26] Rubei, R., Moussaid, A., Di Sipio, C., & Di Ruscio, D. (2025). Prompt engineering and its implications on the energy consumption of Large Language Models. *arXiv preprint arXiv:2501.05899.* https://doi.org/10.48550/arXiv.2501.05899

[27] Hoy, M. B. (2018). Alexa, Siri, Cortana, and more: an introduction to voice assistants. *Medical reference services quarterly*, *37*(1), 81-88. https://doi.org/10.1080/02763869.2018.1404391

[28] Bellegarda, J. R. (2013, August). Large-scale personal assistant technology deployment: the siri experience. In *INTERSPEECH* (pp. 2029-2033). https://doi.org/10.21437/Interspeech.2013-481

[29] Chan, S., Fu, S., Li, J., Yao, B., Desai, S., Prpa, M., & Wang, D. (2024). Human and llm-based voice assistant interaction: An analytical framework for user verbal and nonverbal behaviors. *arXiv preprint arXiv:2408.16465*. https://doi.org/10.48550/arXiv.2408.16465

[30] Yu, D., & Deng, L. (2016). *Automatic speech recognition* (Vol. 1). Berlin: Springer. https://doi.org/10.1007/978-1-4471-5779-3

[31] Tan, X., Qin, T., Soong, F., & Liu, T. Y. (2021). A survey on neural speech synthesis. *arXiv preprint arXiv:2106.15561*. https://doi.org/10.48550/arXiv.2106.15561

[32] Lu, J., Wu, D., Mao, M., Wang, W., & Zhang, G. (2015). Recommender system application developments: a survey. *Decision support systems*, *74*, 12-32. https://doi.org/10.1016/j.dss.2015.03.008

[33] Lü, L., Medo, M., Yeung, C. H., Zhang, Y. C., Zhang, Z. K., & Zhou, T. (2012). Recommender systems. *Physics reports*, *519*(1), 1-49. https://doi.org/10.1016/j.physrep.2012.02.006

[34] Adomavicius, G., & Tuzhilin, A. (2010). Context-aware recommender systems. In *Recommender systems handbook* (pp. 217-253). Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-85820-3_7

[35] Yang, Z., Khatibi, E., Nagesh, N., Abbasian, M., Azimi, I., Jain, R., & Rahmani, A. M. (2024). ChatDiet: Empowering personalized nutrition-oriented food recommender chatbots through an LLM-augmented framework. *Smart Health*, *32*, 100465. https://doi.org/10.48550/arXiv.2403.00781

[36] Zebra Technologies. (n.d.). *PS20 personal shopper specification sheet.* Retrieved from https://www.zebra.com/us/en/products/spec-sheets/mobile-computers/handheld/ps20.html

[37] Supersmart Ltd. (2024, February 13). *Revolutionizing Retail: Osher-Ad's Leap Forward with SuperSmart Technology*. Retrieved from https://supersmart.me/revolutionizing-retail-osher-ads-leap-forward-with-supersmart-technology/

[38] SuperSmart Ltd. (n.d.). *Solutions*. Retrieved from https://supersmart.me/solutions/

[39] Cust2Mate Ltd. (n.d.). *Media.* Retrieved from https://cust2mate.com/media/

[40] Cust2Mate Ltd. (n.d.). *Yochananof case study*. Cust2Mate. Retrieved from https://cust2mate.com/yochananof-case-study/

[41] Cust2Mate Ltd. (n.d.). *Platform.* Cust2Mate. Retrieved from https://cust2mate.com/platform/

[42] Shopic Inc. (n.d.). *Platform.* Shopic. Retrieved from https://www.shopic.co/platform/

[43] OpenAI. (n.d.). *Function calling in the OpenAI API.* Retrieved from https://platform.openai.com/docs/guides/function-calling/

[44] Google AI for Developers. (n.d.). *Function calling with the Gemini API.* Retrieved from https://ai.google.dev/gemini-api/docs/function-calling

[45] Murtaza, F. (2021, December 3). *Microcontrollers vs. single-board computers: What's the difference?* MakeUseOf. https://www.makeuseof.com/microcontrollers-single-board-computer-differences/

[46] Espressif Systems. (2025). *ESP32 Series Datasheet (Version 4.9)*. Retrieved from https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[47] Raspberry Pi Ltd. (2025, January). *Raspberry Pi 5 product brief* (Product Brief). Retrieved from https://datasheets.raspberrypi.com/rpi5/raspberry-pi-5-product-brief.pdf

[48] Weber, B. (2023, November 18). *Raspberry Pi 5 review*. bret.dk. Retrieved from https://bret.dk/raspberry-pi-5-review/

[49] Raspberry Pi Foundation. (n.d.). *Getting started with your Raspberry Pi.* In Raspberry Pi Documentation. Retrieved from https://www.raspberrypi.com/documentation/computers/getting-started.html

[50] Raspberry Pi Foundation. (n.d.). *Raspberry Pi OS* (in *Raspberry Pi Documentation*). Retrieved from https://www.raspberrypi.com/documentation/computers/os.html

[51] Yankulov, K. (2025). *Vapi Voice AI review: A comprehensive analysis and key takeaways*. Retrieved from https://softailed.com/blog/vapi-review

[52] Artificial Analysis. (n.d.). *Comparison of models: Intelligence, performance & price analysis*. Retrieved from https://artificialanalysis.ai/models

[53] React (Meta Platforms, Inc.). (n.d.). *Describing the UI.* In React Documentation. Retrieved from https://react.dev/learn/describing-the-ui

[54] systemd. (n.d.). *System and service manager*. Retrieved from https://systemd.io/

[55] Google. (n.d.). *Routing options: First solution strategy.* In Google OR-Tools documentation. Retrieved from https://developers.google.com/optimization/routing/routing_options#first_solution_strategy

[56] Python-evdev. (n.d). *nputDevice.grab()* [Method documentation]. In *python-evdev API reference.* Retrieved from https://python-evdev.readthedocs.io/en/latest/apidoc.html#evdev.device.InputDevice.grab

[57] OpenAI. *Smart cart engineering cost estimates [ChatGPT conversation].* https://chatgpt.com/share/68866a1e-ba10-8001-b303-5c0125ab8c8b