Assignment #8 - Automated Smart Shopping Cart

Software Systems Architecture

Alexandre Ferreira Nunes
André Correia da Costa
André Filipe Garcez Moreira de Sousa
Daniel José Mendes Rodrigues
Gonçalo da Costa Sequeira Pinto

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

April 2024

Contents

1 Introduction

This report presents the architectural design of the Automated Smart Shopping Cart, a system tailored to revolutionize the grocery shopping experience. Traditional grocery shopping often involves long checkout lines, manual scanning of items, and cumbersome payment processes, leading to frustration and inefficiency for customers. In response to these challenges, the Automated Smart Shopping Cart offers a streamlined and automated approach to grocery shopping.

By leveraging advanced technology, including real-time item detection and seamless payment integration, the system enables users to add and remove items from their cart while keeping track of the total price. Additionally, personalized discounts and offers are provided through Customer memberships, enhancing the overall shopping experience.

Throughout this report, we will delve into the Automated Smart Shopping Cart's architecture, examining its logical, process, physical, deployment, and use-case views. We will elucidate the underlying components, interactions, and use cases, shedding light on the system's design principles and key architectural decisions.

2 Automated Smart Shopping Cart's Architecture

2.1 Logical view

The logical view of the Automated Smart Shopping Cart system (**Figure 1**) can be decomposed into several key components, each responsible for different functionalities. These components interact to provide a seamless shopping experience for customers while ensuring efficient management for the store.

2.1.1 User Interface

The User Interface (UI) component encompasses the interfaces through which users interact with the system. This includes the mobile application for customers, the display device attached to the shopping cart for non-app users, and any other interfaces used for input or output. We anticipate the usage of the following components:

- **Device Pairing:** In this component, the users will be able to pair their mobile phones with the shopping cart's device using wireless technology such as Bluetooth. The users will choose their mobile phone from the near devices list.
- **Product List Display:** This is the component that is responsible for keeping the users informed of the items that were added to their shopping carts, as well as the current amount due.

- **Item Lookup Interface:** In this component, users can scroll through a menu that is divided into logical sections (bakery products, fish, meat, etc) so that they can learn the position of the different products they are interested in.
- **Item Lookup Result:** With this component, users can visualize the aisle in which the products that they searched for are present.
- **Checkout Interface:** This is the component where users can input their payment information and confirm the list of products that they are about to purchase.
- **Admin Page:** With this component, the store administrators can change the item's prices and the discounts associated with them. This component communicates directly with the Cloud Component to apply the changes. It is worth noting that the administrators cannot apply changes to the products' prices and discounts if those changes would have an immediate impact, to avoid price inconsistencies between the time where a client places a product in the basket and when he performs the checkout.

2.1.2 Cart's Device Components

The Cart's device serves as a crucial interface between the physical shopping cart and the Automated Smart Shopping Cart system. It integrates various components to detect items and facilitate transactions directly from the cart. The following are the key software components present in the Cart Device:

- **Auth:** This is the component responsible for receiving the users' data from the Device Pairing module and checking if the information is valid. Furthermore, it communicates with the Cloud service to fetch the coupons that are associated with the users' account and then routes the valid coupons to the Cart Manager, so that the displayed amount due already includes the discounts.
- **Item Detector:** Hardware component developed by the hardware team that informs the Cart Manager of what action occurred and which product was involved. With this goal, it should send a message to the Cart Manager with an operation type ("ADD" or "REMOVE"), the product ID (a unique number that allows the Cart Manager to identify the product), and the quantity of the product.
- **Item Manager:** This component knows where every product of that particular store is located, so it is used by the User Interface components to look up the location of certain products whose locations are not known to the user.
- **Cart Manager:** This is the main component of the Automated Smart Shopping Cart system.

Firstly, it is used by the Product List Display component to show the user the current state of their shopping task.

Secondly, it receives information from the Auth and Item Detector modules to know about the coupons of the user and the changes to his shopping cart, respectively.

Furthermore, it communicates with the Alarm Manager to stop malicious (or distracted) users from leaving the shop without paying for their purchases.

Moreover, when it is time to checkout, it sends all the coupons and the shopping list to the Checkout component, so that the user can pay for the products he bought, as well as apply the discounts.

- **Alarm Manager:** When the alarm's sensor detects that a shopping cart is approaching the exit, it communicates with the shopping cart's device to ask the Cart Manager if the shopping cart is authorized to leave the shop (in other words, it asks if the payment process was completed). If this is not the case, the alarm is triggered.
- **Checkout:** This component receives the shopping list from the Cart Manager and uses the Checkout Interface module to decide what is the payment method and, with that information, communicate with the Payment Service to finalize the process of buying products from the store.
- **Price Enforcer:** The Price Enforcer component is a sub-routine that is run daily in each store to make sure that all the shopping carts' devices have the correct price information. This is required to make sure that the prices displayed on the screen of the shopping cart are up-to-date and that no administrator of the store changed the price of the item during the shopping task of the user, originating an inconsistency that would affect the customer's trust in the company.

2.1.3 Remote Services

The Automated Smart Shopping Cart system relies on various remote services hosted on cloud platforms and backend servers to support its functionalities seamlessly. These remote services include the cloud infrastructure for managing coupons, product prices, discounts, and inventory across all stores, as well as dedicated services for inventory management and payment processing. The following are the key remote services integral to the system:

- **Cloud Infrastructure:** The Cloud Infrastructure serves as the central hub for managing critical data and services required for the operation of the Automated Smart Shopping Cart system. It hosts databases, application servers, and web services responsible for managing coupons, product prices, discounts, and inventory across all stores within the retail chain. The cloud infrastructure ensures scalability, reliability, and accessibility of data and services from anywhere, enabling real-time updates and synchronization across the entire ecosystem.

- **Inventory Manager:** The Inventory Manager is a dedicated service responsible for managing inventory for each store within the retail chain. It interfaces with the cloud infrastructure to update the real-time inventory data of that particular store. Furthermore, this component also requires interaction with the Checkout, because when a user checks out his purchases, the list of products that were removed from the store is sent to this component, to update the inventory data.

- **Payment Service:** The Payment Service handles payment processing for each user, facilitating secure and seamless transactions across multiple payment methods. It interfaces with external payment gateways and financial institutions to authorize and process payments, verify user credentials, and update transaction records in real time. The Payment Service supports various payment methods, including credit/debit cards, mobile wallets, and digital payment platforms, catering to diverse customer preferences and ensuring a frictionless checkout experience.

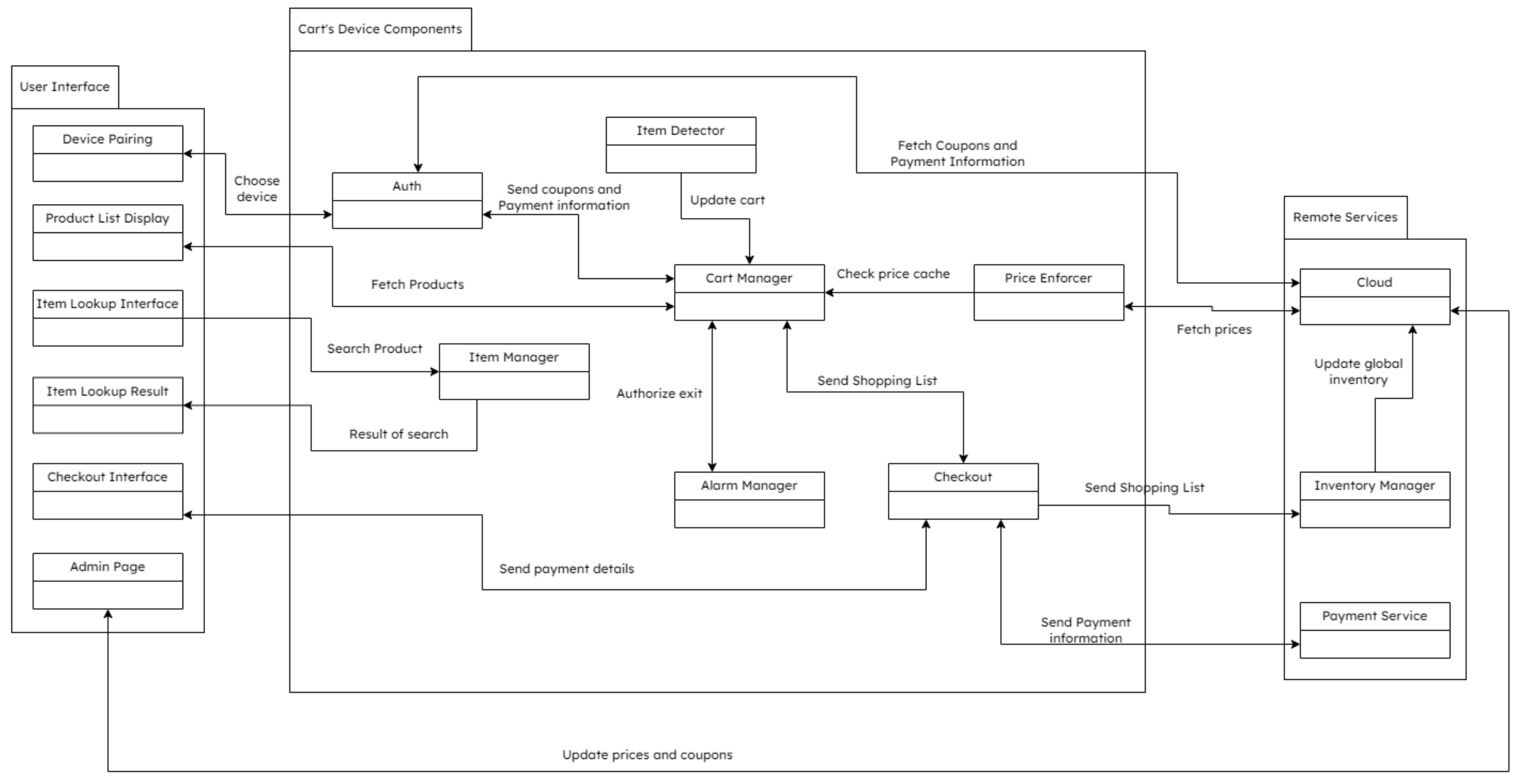


Figure 1: Logical view

2.2 Process view

The process view depicted in **Figures 2 and 3** shows the interactions between different parts of the Automated Smart Shopping Cart system. The process involved interactions between **11** distinct participants: **User**, **Terminal**, **Price Enforcer**, **Cloud**, **Item Manager**, **Payment Service**, **Checkout**, **Auth**, **Cart Manager**, **Inventory Manager** and **Alarm Manager**. In this system, the User is the main actor, the Terminal is the User Interface (the tablet in the shopping cart), the Cloud englobes the shop's database as well as other shop services like the admin web page, and every other participant is a **backend module**, each with a different functionality.

The Price Enforcer runs a routine every 24 hours that checks the version of the prices in the shopping cart and validates them with the price version in the cloud. If they don't match, the shopping cart is updated with the new prices.

Once a user gets a cart, he is asked to connect his phone (via Bluetooth) to sync his account and get coupon information and preferred payment methods. If he doesn't have a phone or doesn't want to connect, the terminal works the same way, but he won't get any discounts and has to input his payment information at checkout.

The item manager is responsible for the item lookup feature. The user can scroll through a menu that is divided into logical sections (bakery products, fish, meat, etc) so that he learns the position of the product he is interested in.

Every time the terminal detects a change in the items, it determines what was added/removed and calls a Cart Manager function to log that information.

The alarm manager grants the cart doesn't leave the store before the user pays for the products. If he tries that, he gets asked to perform a checkout. If he doesn't do it, the module sends a signal to the shopping cart to block the wheels and contacts the authorities. On the other hand, if he proceeds to perform the checkout, the Checkout module gets called by the terminal, which provides coupon and payment information as arguments. It communicates with the Cart Manager to get the price of the items, then it calculates the total price by applying the coupons and proceeding with the payment using the Payment Service. After it is done, it calls the Inventory Manager to update the inventory in the shop's database.

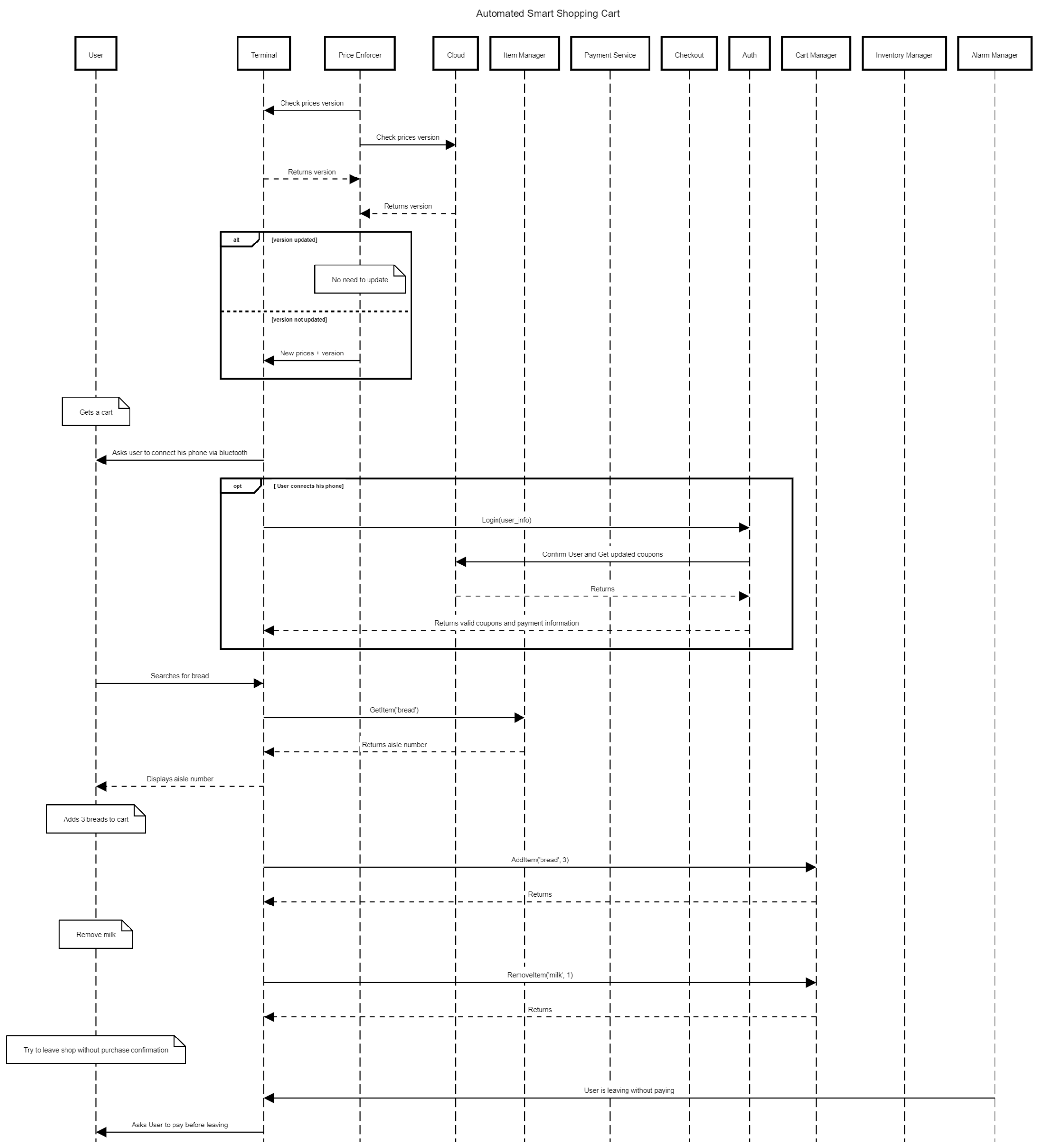


Figure 2: Process View (Part 1)

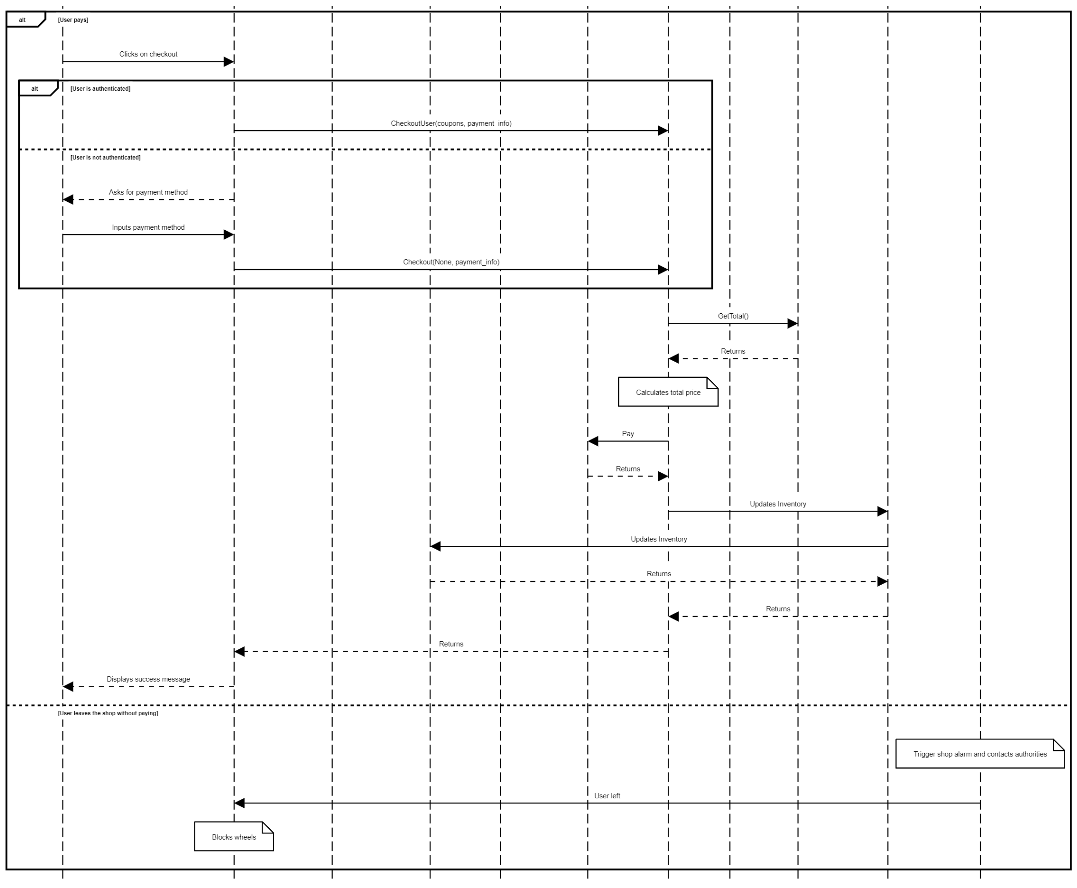


Figure 3: Process View (Part 2)

2.3 Physical View

The previously mentioned components will run on four types of hardware: in the shopping carts, in the store, on users' mobile phones, and in the cloud. In the hardware of the shopping carts, two devices will be installed: an item recognizer, responsible for detecting additions or removals of products in the cart and communicating these changes to the terminal, which constitutes the second hardware present in the shopping carts. The terminal will be a central piece in the user experience within the store, responsible for executing the software related to purchases. In addition to the terminal, the software will also run on the customer's mobile application and the alarm devices in the stores. Finally, in the cloud, there will be two proxies, one to redirect requests to the payment servers and another to redirect requests to the server that will execute the business logic. The latter will also be connected to databases that store data about clients, products, and other things. Figure 4 provides a graphic depiction of the described information.

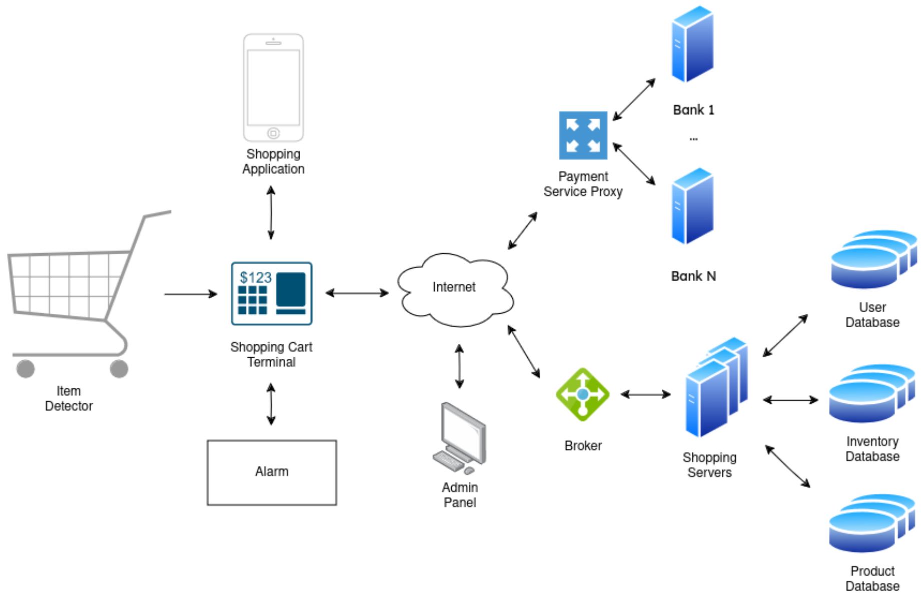
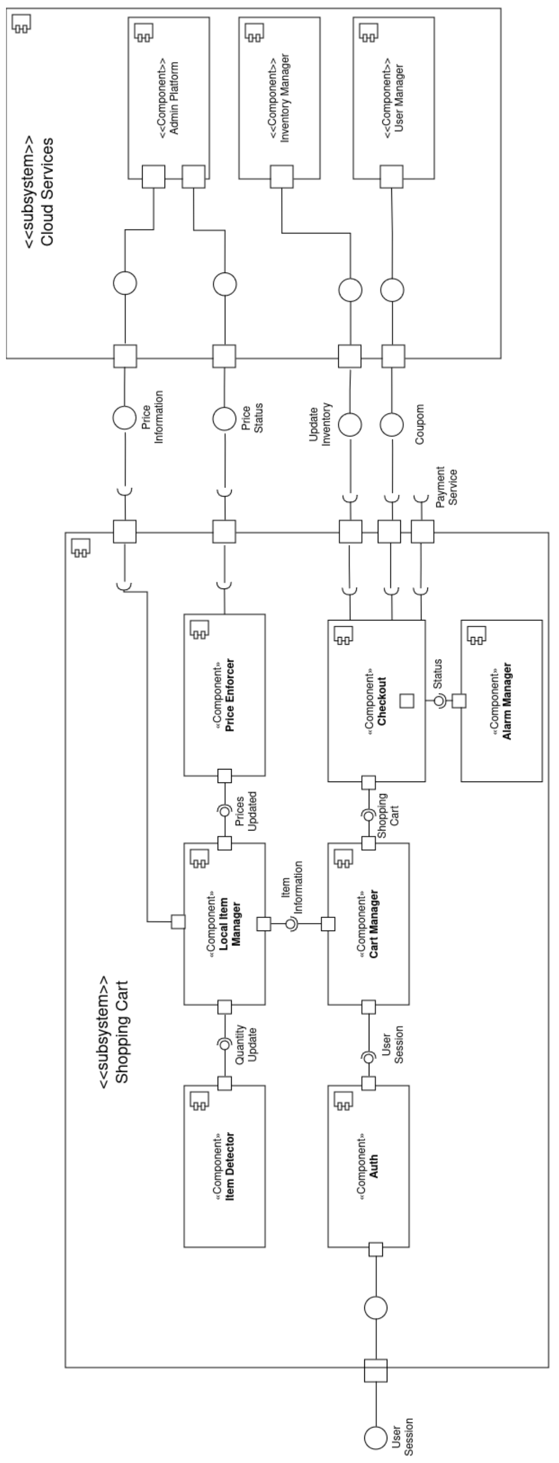


Figure 4: Physical View

2.4 Development View

The Development View is represented by a component diagram in **Figure 5**.



<<subsystem>> Cloud Services
<<Component>> Admin Platform
<<Component>> Inventory Manager
<<Component>> User Manager
Price Information
Price Status
Update Inventory
Coupom
Payment Service
<<subsystem>> Shopping Cart
«Component» Price Enforcer
«Component» Checkout
Status
«Component» Alarm Manager
Prices Updated
Shopping Cart
«Component» Local Item Manager
Item Information
«Component» Cart Manager
Quantity Update
User Session
«Component» Item Detector
«Component» Auth
User Session

Figure 5: Development View

2.5 Use-case view

In the Use Cases view, the various scenarios depicting user interactions with the Automated Smart Shopping Cart system are outlined.

2.5.1 Main Use Cases

- User add/remove an item
- User adds items to the cart and leaves after paying.
- User adds items to the cart and leaves without paying, which triggers the alarm.
- User connects to the cart.
- User buys items with coupons.
- User looks up a product by its category.
- Two users try to use the same coupons at the same time.

2.5.2 Use-case Diagram

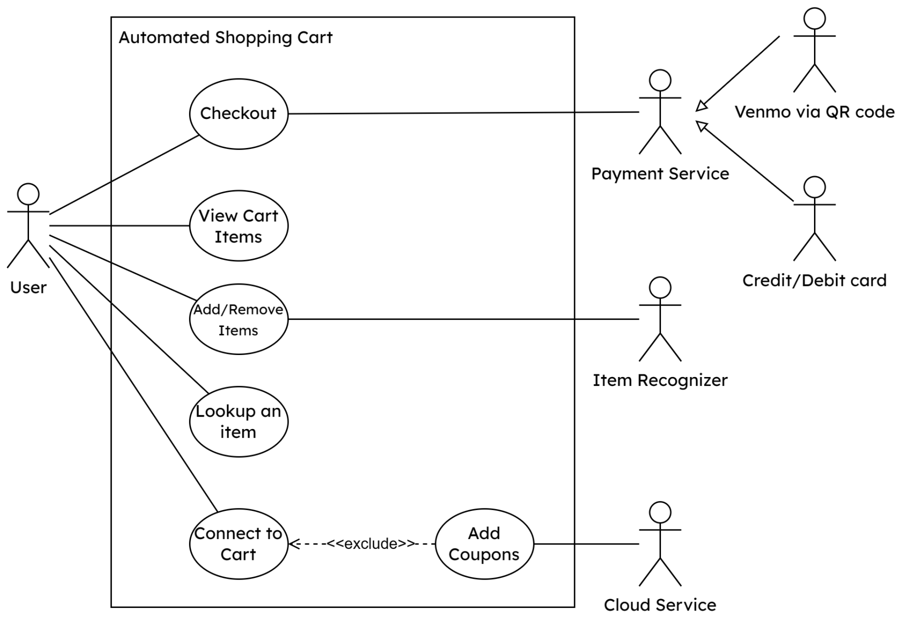


Figure 6: Use-case Diagram

When starting a shopping session, the initial step involves the user connecting to their cart, in case they have the app installed. This connection is facilitated through Bluetooth, which would enable the retrieval of coupons from the cloud once established.

One of the simpler use cases would be adding or removing items, which is managed by external hardware referred to as the "Item Recognizer." This hardware is equipped with specialized small software whose sole purpose is to communicate with the Terminal, where our primary software components reside. The communication would consist of simple messages indicating the ID of the item, the operation (add or remove), and the quantity of the product.

The Terminal, acting as the hub, stores and handles all cart operations, displaying in a user interface the details such as current items, and total price, and offering a lookup feature to locate items within the store. It's worth noting that these operations are independent of cloud communication; the cart itself holds all necessary shopping data, periodically synchronizing it with the cloud during off-hours.

During the checkout phase, the Terminal communicates with the cloud service to validate coupon usage, confirming the final total. Subsequently, the user proceeds to payment, which can be completed either via a QR code scan or using a credit/debit card through dedicated hardware integrated into the cart.

3 Key Use Cases

In this section, it will be discussed in more detail the two key use cases.

3.1 Adding an item to the cart

Once the terminal notices a difference in the items it is carrying in the shopping cart, it uses the **Item Recognizer**. When he knows what was added, in this case, was a bottle of water, it calls the Cart Manager's AddItem function, providing the item identifier and the quantity that was added. After returning successfully, the terminal shows the user an animation of the item being added to the cart, so he is aware of what is happening in the cart at every moment of his shopping experience. This process is better explained in **Figure 7**

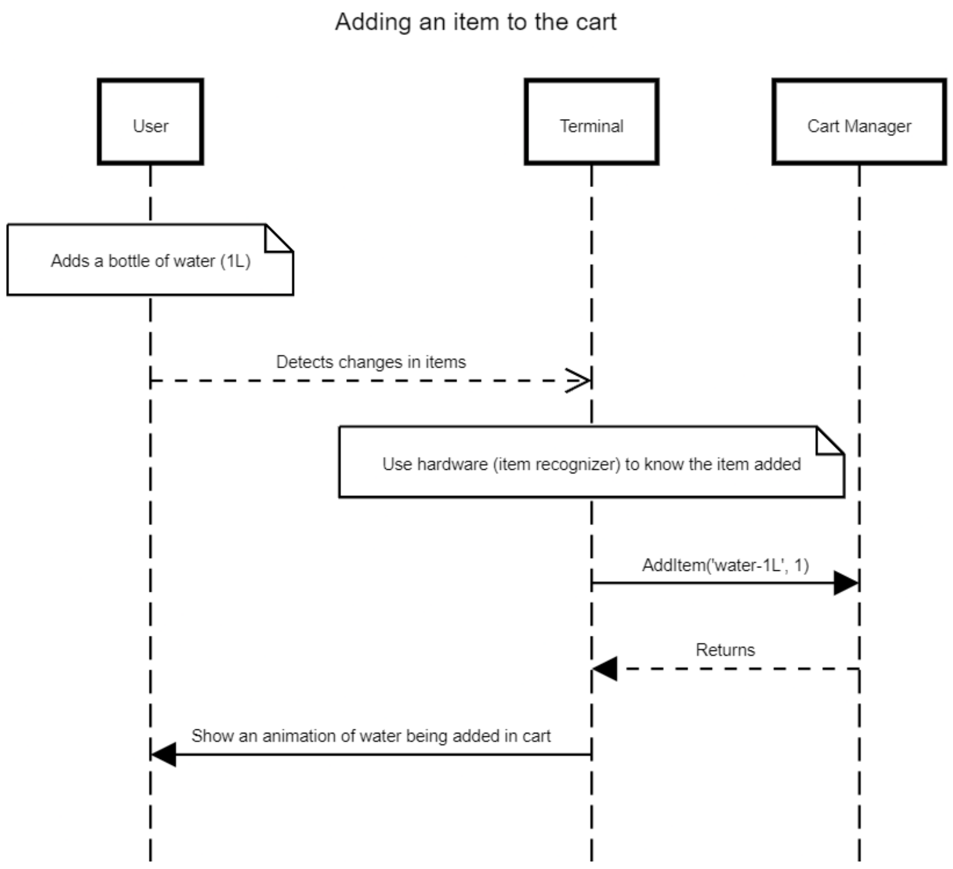

Adding an item to the cart
User
Terminal
Cart Manager
Adds a bottle of water (1L)
Detects changes in items
Use hardware (item recognizer) to know the item added
AddItem('water-1L', 1)
Returns
Show an animation of water being added in cart

Figure 7: Use Case 1

3.2 Walking out of the store (checkout)

The normal checkout can be divided into two use cases: **Authenticated Checkout** and **Guest Checkout**. However, **Figure 8** shows both. Firstly, the user clicks the checkout button on the terminal or the phone (if he is authenticated). Then if he is authenticated, the terminal calls the checkout function directly, because he already has every information needed (coupons and payment). If he is not authenticated, the terminal asks him to input payment information before calling the checkout function. After that, the Checkout module gets the total value of the items using the Cart Manager's GetTotal function and then applies the coupons if there are any. Once everything is ready, it calls the Payment Service to continue with the payment process, using the preferred payment method. Finally, it calls the inventory manager to remove the items bought in the cloud database.

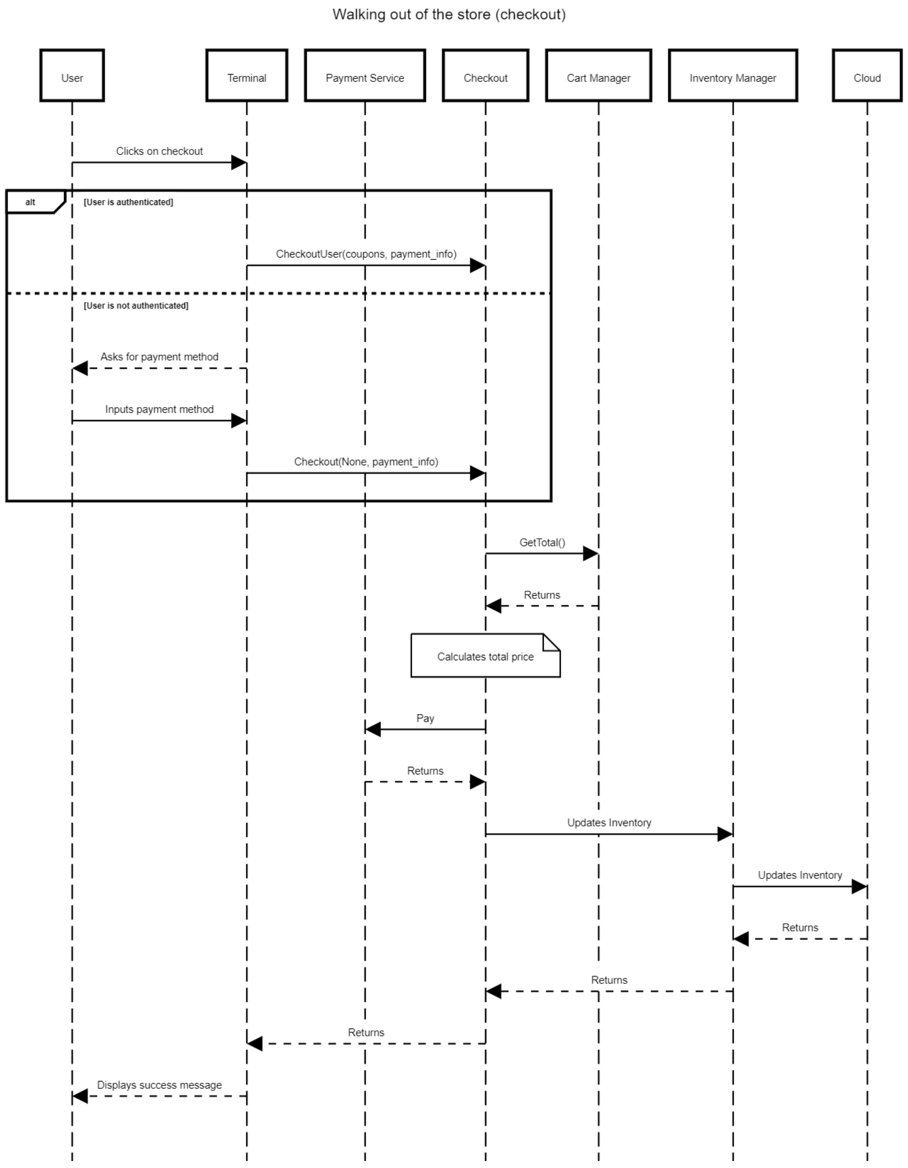


Figure 8: Use Case 2

4 Architectural Patterns

In the architecture designed for the Automated Smart Shopping Cart, several software architectural patterns are employed to enhance its robustness, scalability, and maintainability.

4.1 Component-Based Architecture

The system employs a component-based architecture, where each functional unit is implemented as a separate component. These components interact with each other to fulfill the system's requirements. This architecture promotes modularity, reusability, and maintainability by encapsulating specific functionalities within individual components.

4.2 Event-Driven Architecture

The system utilizes an event-driven architecture to facilitate communication and coordination between components. Events are generated by various actions and triggers within the system (such as adding or removing items from the cart, or detecting unauthorized exits), and these events drive the behavior and responses of other components. This architecture enhances flexibility, scalability, and responsiveness by enabling asynchronous communication and decoupling between system modules.

4.3 Client-Server Architecture

The system incorporates client-server interaction, where client components (such as the mobile application and Shopping cart unit) communicate with backend server components to retrieve data, process requests, and perform operations. This architecture allows for centralized management of critical functionalities (such as inventory management and payment processing) and ensures consistency and reliability across the system.

4.4 Broker Architecture

The Broker Pattern is employed to facilitate communication and coordination between the client-side (UI) and the server-side components of the system. The broker acts as an intermediary, managing and routing messages between clients and servers. It functions as a load balancer, distributing incoming requests among multiple server instances to ensure efficient resource utilization and scalability.

4.5 Publish-Subscribe Pattern

The broker between client-side and server-side implements a pub/sub (publish/subscribe) communication model, where software components publish information (events) and subscribe to relevant topics of interest. This pub/sub model helps keep the communication traffic clean and organized, allowing components to communicate asynchronously only when updates are made. For instance, when updates to the products are required, the shopping carts terminals are notified about since they are subscribing all the products information.

4.6 Shared Repository Pattern

Automated Smart Shopping Cart employs the Shared Repository pattern for managing the store's inventory. The Inventory Storage package acts as a centralized repository accessible by multiple components, fostering data consistency and integrity throughout the system.

5 Quality Attributes

This section presents the Non-Functional Attributes of this system, ordered by the Importance parameter and accompanied by its meaning to the system and an Acceptance Level.

5.1 Reliability

The system's capacity to perform specific operations under some conditions for some time.

Acceptance Level:
The system must not fail during normal utilization. It must accurately detect products and identify instances of non-payment, triggering the alarm in case of attempted theft, while also avoiding false alarms.

Importance:
Very High. A failure in the system would affect the whole belief of the customer in it.

Architectural Impact:
In terms of the server side, the architecture presents a broker with load balance mechanisms to distribute and accommodate several servers running to handle different fluctuations of workload, preventing failures on this part. Also, the system's modularity prevents different components from affecting other parts of the system keeping it operational and fault-tolerant.

5.2 Availability

The system's capacity to be operational and accessible when required for use.

Acceptance Level:
The system must be 100% available during open hours. Mainly the shopping

cart software and hardware.

Importance:
Very High. Since the system only provides this way to shop, ensuring availability is a high demand so customers can go shopping.

Architectural Impact:
As mentioned in the Reliability, the system's modularity and broker allow it to handle high loads and fluctuations in workload without compromising availability.

5.3 Security

The degree to which the system defends itself from malicious users and protects its customers' data.

Acceptance Level:
The system will employ encryption protocols, access controls, and regular security audits to safeguard customer data against unauthorized access and breaches.

Importance:
High. The system is dealing with customers' sensitive information, like payment information, so it must be robust against malicious Users. A fault in this attribute would create a lack of confidence in the system.

Architectural Impact:
Besides the encryption protocols, the architecture presents an Authentication Component, used to sync the User Application with the Shopping Cart functionality, so the User can pursue discounts/coupon validation and payment process through the application.

5.4 Performance

The ability of the system to perform its functions within a specified time and efficiently use its resources under certain conditions.

Acceptance Level:
The system must ensure that customers are not kept waiting for more than 10 seconds during any interaction. It should maintain consistent performance levels, even during peak hours when the number of customers in the supermarket may exceed 200.

Importance:
Medium-High. The system is expected to experience increased traffic during peak periods, and customers in these situations are likely to be in a hurry. Therefore, ensuring optimal performance is crucial to meet user expectations

and maintain satisfaction with the shopping experience.

Architectural Impact:
Within this Architecture design, a Message Broker is used to promote the distribution of workload and scalability of several servers, to deal with the bigger fluctuation of requests during peak hours.

5.5 Maintainability

The system's effectiveness and efficiency in being modified, corrected, or adapted to meet the specified system requirements.

Acceptance Level:
The system presents this modular architecture, preventing high dependency between software and hardware components, although they still communicate. System updates in general should be made on close time.

Importance:
Medium. Multiple updates are required to products' stock and prices. Since we have a main component of the system, the shopping cart, its maintainability can be critical to the system.

Architectural Impact:
By isolating functionalities within distinct components, the architecture promotes maintainability. Developers can focus on modifying specific components without needing to delve into the intricacies of the entire system. This reduces the risk of introducing bugs during maintenance activities.

5.6 Scalability

The ability to adjust the system's capacity to accommodate fluctuations in workloads.

Acceptance Level:
The system must be able to add/remove carts during open hours without affecting other quality attributes and without stopping the entire system.

Importance:
Medium. Although it would not happen a lot, it could happen and it is important that the system does not stop or is affected by this case scenario.

Architectural Impact:
The modularity of the system promotes scalability in terms of adding new components and the broker also brings the possibility to add more servers to deal with more customers' demand without affecting the rest of the system.

5.7 Usability

The degree to which a system brings satisfaction to a user. It is measured in factors such as intuitiveness, efficiency, learnability, and user satisfaction.

Acceptance Level:
The shopping cart interface and the app should feature a cohesive and user-friendly design to enhance ease of use. The shopping cart interface should be touch-responsive and visually informative, displaying product details and their location within the store.

Importance:
Medium. Since the system interacts directly with customers, the UIs must be easy to understand, since our target audience is very omnibus.

Architectural Impact:
This attribute does not have an impact on the architecture, besides the incorporation of similar user-friendly designs of the Shopping Cart UI and the User Mobile Application.

5.8 Portability

The ease with which a software system can be transferred or adapted to different hardware or software environments.

Acceptance Level:
The mobile application should be compatible with both iOS and Android operating systems, ensuring accessibility to a broad range of users who utilize either platform.

Importance:
Medium-Low. While the system primarily relies on the shopping cart software, ensuring its smooth operation is paramount. Compatibility with mobile applications serves as a secondary consideration, providing added flexibility for users who prefer alternative access methods.

Architectural Impact:
While the system primarily relies on the shopping cart software, compatibility with mobile applications is ensured by designing the mobile application to be compatible with both iOS and Android operating systems. This does not have an impact on the architecture, since it does not separate iOS from Android.

5.9 Extensibility

The ability of the system to easily accommodate the addition of new features or functionality without requiring significant modification to its existing structure.

Acceptance Level:
The system architecture should be designed with modularity and flexibility to allow for the seamless integration of new features if required.

Importance:
Low. This system would not require many additions or new features.

Architectural Impact:
The Independence between components promotes the addition of new features by the developer team, without interfering with the rest of the system components' code.