Faculty of Engineering of the University of Porto

Master in Informatics and Computing Engineering

Software Systems Architecture

Automated Smart Shopping Cart

Homework 05

Team 32

José Francisco Veiga up202108753@up.pt

Marco Vilas Boas up202108774@up.pt

Pedro Lima up202108806@up.pt

Pedro Januário up202108768@up.pt

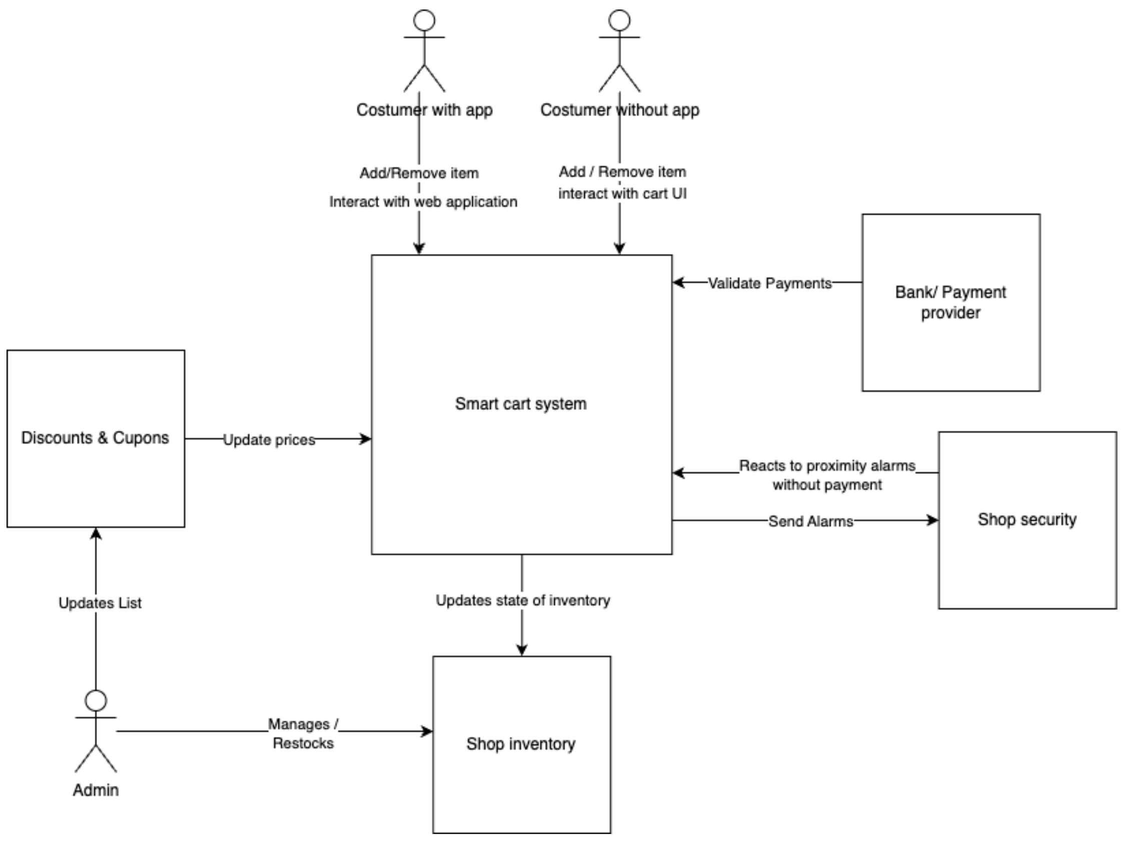
Pedro Marcelino up202108754@up.pt

April 2025

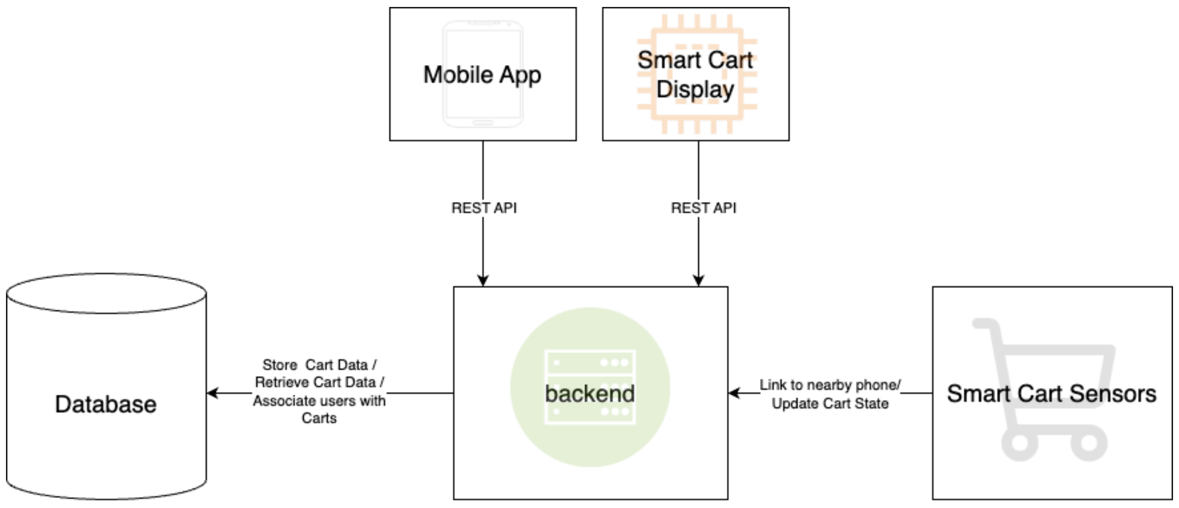
Introduction

This document addresses and details a possible architecture for an **Automated Smart Shopping Cart**. The system is intended to aid the shopping process, sensing when items are put in or taken out carts, automatically billing customers when they checkout.

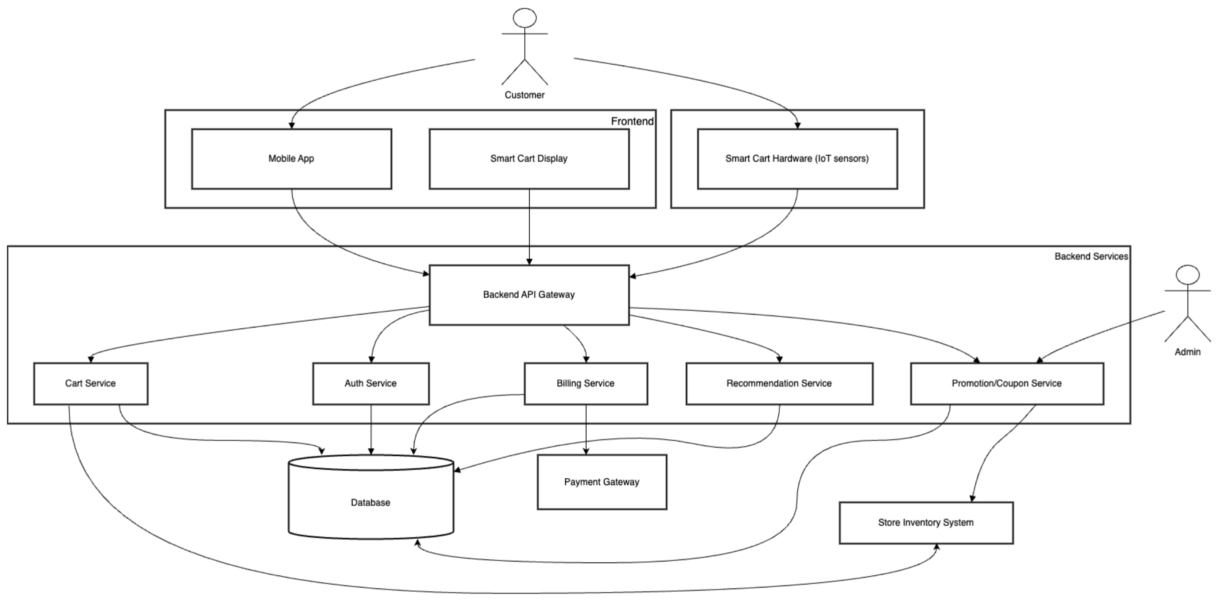
Context Diagram



Containers Diagrams


Mobile App
Smart Cart Display
REST API
REST API
Database
Store Cart Data / Retrieve Cart Data / Associate users with Carts
backend
Link to nearby phone/ Update Cart State
Smart Cart Sensors

Component Diagrams



Architecture Description

Our architecture of the **Automated Smart Shopping Cart** system adopts a modular service-oriented design, structured across 3 main layers: **Frontend**, **Backend** and **Data**.

As shown in the diagrams above, the main components are:

Frontend Layer

- **Mobile App**: this is the main interface for customers to see the running total, current cart items, apply discounts or coupons, and proceed to checkout.
- **Smart Cart Display**: a screen embedded into the shopping cart for users who do not have the mobile app. It mirrors the core functionality of the app and also includes additional features, such as a slot for card payment.
- **Smart Cart Hardware (IoT Sensors)**: these are sensors that detect when items are added or removed from the cart, and also help associating the cart with the customer (via Bluetooth or similar).

Backend Layer

A centralized **Backend API Gateway** receives and routes requests from the Frontend to several specialized microservices, which are:

- **Cart Service**: manages real-time updates to the contents of the customer's cart.
- **Auth Service**: handles authentication and authorization for customers and respective memberships.
- **Billing Service**: calculates the total amount, applies discounts and/or coupons and manages coordination with payment systems.

- **Recommendation Service**: responsible for understanding, processing and learning user history and behavior, so that it suggests products effectively targeted to that user.
- **Promotion/Coupon Service**: manages prices, discounts and electronic coupons for products and users.

Data layer

- **Central Database**: stores all the relevant persistent data necessary for the system, including users, products, transactions and carts.
- **Payment Gateway**: connects with external payment providers, such as credit/debit card, Venmo, wallets, etc. to finalize and process transactions.
- **Store Inventory System**: this is an internal system to manage stock levels. It's integrated with the backend for live synchronization.

Architectural Decisions

Below, we explain some of these main decisions, and associate each with some of those quality attributes:

Layered Architecture with API Gateway

Separating the system into frontend, backend and data layers improves **modularity** and supports **independent scaling**. The API Gateway acts as the single point of entry, simplifying external communication and improving **security** and **observability**.

Microservices-based Backend

By assigning each core functionality to a separate service, the system gains **flexibility**, **independent deployability**, and **fault isolation**. This modularity supports easier future optimizations and adding new features.

IoT Cart Hardware

Using smart sensors in carts automates the detection of items and user presence, improving **efficiency** and eliminating friction at checkout. This supports the core use case of **line-free shopping**.

Dual Interface - Mobile App & Cart Display

Implementing both a mobile app and a physical display ensures **universal accessibility**, supporting users who do not have either smartphones or the app installed. This improves **usability**.

Secure checkout with User Validation

Requiring a PIN or biometric confirmation at checkout guarantees **security** and prevents fraud. If the person pushing the cart is not the registered user, the system can re-authenticate quickly before opening the exit door.

Inventory Integration

By designing the architecture to be compatible with the store's inventory system, the solution becomes **extensible**, ensuring long-term **maintainability** and growth.

Quality Attributes

For this system, some quality attributes are of utmost importance and must always be present:

- **Interaction Capability**:
    - **Inclusivity**: The system must be very easy to use, since it will be used by all kinds of people, including those who are not very comfortable with technology.
    - **Learnability** and **Operability**: The system is designed to follow a simple two-step process that most people already use when shopping: you put items into the cart, and then you pay before leaving. The only difference is that now, you pay directly on the cart or on the mobile app, instead of going to the cashier.
    - **Recognizability**, **User Engagement**, and **Self-Descriptiveness**: It must be very clear to the user what to do, just follow the two steps mentioned above.
- **Functional Suitability**:
    - **Functional Appropriateness**: The goal of the system is to make the shopping experience easier. In the whole process, no steps were added, only the unnecessary ones were removed, so there are no new things for the user to learn.
    - **Functional Completeness** and **Functional Correctness**: The system will include all the features specified in the requirements, and all features will work as expected.
- **Security**:
    - **Resistance**, **Authenticity**, **Accountability**, **Integrity**, and **Non-Repudiation**: All actions go through the backend system, which helps us make sure these five important security aspects are always guaranteed. Also all items a person has in the cart and buys are also stored in the backend, so there is a safe record of the shopping experience.
    - **Confidentiality**: Another important part of security is keeping information private. For example, a person's shopping history can not be publicly available.

Also important, but less crucial:

- **Performance Efficiency**:

    - **Time Behavior**: The cart should detect products quickly, ideally in less than 3 seconds including binding times, but it doesn’t need to be instantaneous.
    - **Capacity**: Each cart has only a small embedded computer, so the software must use very few resources. However, the backend does not have this limitation. Even with limited resources on the cart, the system must still be able to handle a large number of carts, each with lots of products inside. That is one of the reasons the system is more centralized in the backend, giving less freedom to the software in the carts.
- **Reliability**:
    - **Faultlessness**: The carts should not fail all the time. Even though it’s not a critical system, the carts must be reliable. If a cart does stop working, the customer can ask the staff for a new one. It is acceptable that fewer than 1 out of 100 carts fail.
    - **Fault Tolerance**: Even if one or more carts stop working, the rest of the system must keep working normally.
    - **Availability** and **Recoverability**: The backend system must always be available. If it fails, it should recover so quickly that the users don’t even notice.

Other attributes, still important for the system, are safety, flexibility, maintainability and compatibility, but they are not as essential as the ones listed above.

Code Excerpts

In order to help understand how the architectural components work and interact with each other, we present some (pseudo-)code excerpts for two example operations to be carried out by the system. For each situation, we show the code that runs on each involved part. The necessary security checks are not represented, but in the final product they should be present.

Adding an item to a cart

Smart Cart

Unset
```
// An I/O interrupt should call the detectItem function
function detectedItem (server, productId) {
  server.send("addItem", this.id, productId);
}
```

Backend

```
Unset
// runs continuously
function update () {
  while (true) {
    msg <- getCartMessage();
    if (msg.command == "addItem") {
      addItem(DB, msg.id, msg.productId);
    }
  }
  ...
}

function addItem (DB, cartId, productId) {
  customerId <- DB.query(cartId, "customer");

  DB.insert("cartList", customerId, productId);
  productDetails <- DB.query("inventory", productId);

  appId <- DB.query(cartId, "appAssociated");
  if (appId != null) {
    mobileAppInteface <- mobileApp(appId);
    mobileAppInteface.send("addItem", productDetails);
  }

  displayId <- DB.query(cartId, "displayAssociated");
  displayInteface <- display(displayId);
  displayInteface.sendTo("addItem", productDetails);
}
```

Mobile App

```
Unset
// runs continuously
function update () {
  while (true) {
    msg <- getServerMessage();
    if (msg.command == "addItem") {
      ItemsList.append(msg.productDetails);
      RecommendedItems.update(ItemsList);
    }
  }
  ...
  show(ItemsList)
}
```

Cart Display

```
Unset
// runs continuously
function update () {
  while (true) {
    msg <- getServerMessage();
    if (msg.command == "addItem") {
      ItemsList.append(msg.productDetails)
      RecommendedItems.update(ItemsList);
    }
  }
  ...
  show(ItemsList)
}
```

Walking out of the store (checkout)

Smart Cart

```
Unset
function walkOut (server, productList) {
  server.send("checkout", this.id, productList);
}
```

Backend

```
Unset
// runs continuously
function update () {
  while (true) {
    msg <- getCartMessage();
    if (msg.command == "checkout") {
      walkOut(DB, msg.id, msg.productList);
    }
  }
  ...
}

function walkOut (DB, cartId, productList) {
  customerId <- DB.query(cartId, "customer");

  total <- 0;

  foreach (product in productList) {
    total += DB.query(product.id, "value") * product.quant;
    prevQuant <- DB.query(product.id, "stock");
    DB.write(product.id, "stock", prevQuant - prod.quant);
  }

  DB.insert("customerHistory", customerId, productList);

  while (true) {
    msg <- getAppMessage();
    if (msg.command == "checkout" && msg.customerId == customerId) {
      reply("total", total);
      break;
    }
    if (msg.command == "checkout" && msg.cartId == cardId) {
      reply("total", total);
      break;
    }
  }
}
```

Mobile App

Unset

```
function walkOut (storeServer, bankServer, myId) {
  reply <- storeServer.get("checkout", myId);

  if (reply.command == "total") {
    bankServer.pay(reply.total);
  }
}
```

Cart Display

Unset

```
function walkOut (storeServer, bankServer, myId, debitOrCreditCard) {
  reply <- storeServer.get("checkout", myId);
  if (reply.command == "total" && debitOrCreditCard != null) {
    bankServer.pay(reply.total, debitOrCreditCard);
  }
}
```