Projeto de Carrinho de Compras Inteligente
Proposta e Planejamento Inicial

Ângelo Gabriel de Freitas Marques[1], **Domynic Barros Lima**[2], **Juan Pablo Ramos de Oliveira**[3], **Luis Fernando Rodrigues Braga**[4], **Thiago Teixeira Oliveira**[5]

[1] Instituto de Ciências Exatas e Informatica
Pontifícia Universidade Católica de Minas Gerais (PUC MG)
Belo Horizonte, MG – Brazil

Abstract. *The retail purchasing process is marked by efficiency challenges, such as long lines at checkout, customer frustration with ineffective self-service technologies, and a lack of real-time spending control for consumers. This article presents the proposal for a prototype smart shopping cart, aimed at being a pragmatic and low-cost solution. Using an ESP32-based microcontroller, a dedicated barcode scanner, and load cells, management of the shopping list, and real-time visualization of the total amount, also proposing an express checkout flow. The document details the contextualization of the problem, the objectives, a critical review of existing solutions, the development methodology aligned with three sprints, and the limitations of the project.*

Keywords: Smart Shopping Cart, Embedded System, ESP32, Internet of Things, Customer Experience.

Resumo. *O processo de compras no varejo é marcado por desafios de eficiência, como longas filas no checkout, pela frustração do cliente com tecnologias de autoatendimento pouco eficazes e uma falta de controle de gastos em tempo real pelo consumidor. Este artigo apresenta a proposta de um protótipo de carrinho de compras inteligente, focado em ser uma solução pragmática e de baixo custo. Utilizando um microcontrolador baseado em ESP32, um leitor de código de barras dedicado e células de carga, o sistema permitirá a leitura de códigos de barras, a gestão da lista de compras e a visualização do valor total em tempo real, propondo ainda um fluxo de checkout expresso. O documento detalha a contextualização do problema, os objetivos, uma revisão crítica de soluções existentes, a metodologia de desenvolvimento alinhada a três sprints e as limitações do projeto.*

Palavras-chave: Carrinho de Compras Inteligente, Sistema Embarcado, ESP32, Internet das Coisas, Experiência do Cliente.

1. Introdução

O setor de varejo tem passado por uma transformação digital acelerada, buscando soluções que otimizem a jornada de compra e melhorem a eficiência operacional. Um dos principais gargalos em supermercados e lojas de grande porte continua sendo o processo de checkout, que gera grandes filas e, consequentemente, insatisfação nos clientes.

Além disso, os consumidores muitas vezes não possuem uma ferramenta prática para acompanhar o valor total de suas compras em tempo real, o que pode levar a surpresas no caixa. Contudo, as soluções de ponta frequentemente encontram barreiras de viabilidade e aceitação. Por um lado, temos o conceito das lojas autônomas, popularizado pela Amazon Go, que prometem o fim das filas. Recentemente, porém, relatos indicaram que a complexidade da tecnologia exigia intensa supervisão humana para operar, questionando a viabilidade de uma automação completa e de baixo custo em larga escala. Por outro lado, os caixas de autoatendimento, embora difundidos, frequentemente geram frustração devido à alta sensibilidade de seus sistemas de pesagem, que pausam o processo ao menor desvio e exigem constante intervenção de funcionários, anulando o benefício da autonomia.

Adicionalmente, é preciso reconhecer que nenhuma tecnologia é imune a fraudes, como a troca de etiquetas de preço, um problema que transcende a tecnologia e envolve aspectos culturais. Diante disso, nossa hipótese não é criar um sistema utópico e à prova de falhas, mas sim uma ferramenta de baixo custo que visa auxiliar e agilizar o processo de compras existente. A proposta foca em resolver dores reais do consumidor: a falta de controle sobre o valor total da compra em tempo real e a demora no processo de pagamento.

O objetivo geral deste trabalho é projetar e desenvolver um protótipo funcional de um carrinho de compras inteligente que melhore a experiência do cliente, oferecendo transparência e a possibilidade de um checkout mais rápido.

Como objetivos específicos, podemos citar:

- Integrar um microcontrolador NodeMCU V3 a um leitor de código de barras (GM65) e a um sistema de pesagem com células de carga.
- Desenvolver uma interface de usuário intuitiva (simulada em um dispositivo móvel) para gerenciar a lista de compras.
- Propor e simular um fluxo de checkout expresso, onde o cliente poderia realizar um pré-pagamento do valor calculado pelo carrinho, passando por uma verificação final rápida no caixa.
- Validar a viabilidade da solução através de um protótipo físico de baixo custo.

Este artigo está estruturado da seguinte forma: a Seção 2 apresenta uma revisão crítica da literatura e de soluções correlatas. A Seção 3 detalha a metodologia de desenvolvimento e o cronograma. A Seção 4 define o escopo e as limitações do trabalho. Por fim, a Seção 5 aponta para o repositório do projeto.

2. Revisão Bibliográfica

A construção de um sistema de carrinho de compras inteligente tangencia diversas áreas da computação. Esta seção aborda os conceitos fundamentais em Arquitetura de Computadores e Redes de Computadores que sustentam o projeto.

2.1. Arquitetura de Computadores

A base do nosso protótipo é um sistema embarcado. Diferente de computadores de propósito geral, sistemas embarcados são projetados para executar tarefas específicas com restrições de custo, energia e tamanho. O componente central do nosso projeto é

o microcontrolador NodeMCU V3, semelhante ao ESP32, que se destaca por sua arquitetura de System-on-a-Chip (SoC). Sua arquitetura integra processamento dual-core, memória e módulos de conectividade Wi-Fi e Bluetooth, oferecendo um balanço ideal entre custo, consumo de energia e capacidade de processamento para aplicações de IoT [Espressif Systems 2023]. A capacidade de gerenciar múltiplos periféricos de forma concorrente, como a leitura serial do scanner de código de barras, a consulta às células de carga e a comunicação via Wi-Fi, coloca em prova os limites de seu poder computacional, tornando-o um objeto de estudo interessante para esta aplicação e um desafio técnico relevante, que define a fronteira da viabilidade para soluções de baixo custo.

2.2. Redes de Computadores

A funcionalidade de um carrinho inteligente é amplificada quando ele se conecta a uma infraestrutura maior, um conceito central da Internet das Coisas (IoT). O módulo Wi-Fi integrado ao NodeMCU permite que o carrinho se conecte à rede do estabelecimento comercial. Essa conexão pode ser utilizada para diversas finalidades: consultar um banco de dados central para obter preços e informações atualizadas dos produtos [Gubbi et al. 2013], enviar os dados da compra para um sistema de pagamento ao final da jornada, ou até mesmo para coletar dados analíticos sobre o comportamento do consumidor na loja. O uso de protocolos de rede leves, como o MQTT (Message Queuing Telemetry Transport) que é ideal para este tipo de aplicação, garantindo a troca de informações com a infraestrutura da loja de forma eficiente [Kopetz 2011].

2.3. Sistemas Operacionais

O projeto também se relaciona com o campo de Sistemas Operacionais, pois o NodeMCU normalmente opera sobre o FreeRTOS, um sistema operacional de tempo real (RTOS). O FreeRTOS gerencia as múltiplas tarefas do sistema, como a leitura de dados seriais do scanner, o monitoramento das células de carga, a atualização da interface e a comunicação em rede de forma concorrente e previsível, garantindo que o sistema responda de maneira determinística às ações do usuário, garantindo uma experiência de usuário fluida e responsiva [Barry 2010].

2.4. Soluções Correlatas e o Estado da Arte

O estado da arte inclui desde os carrinhos da Caper (agora Instacart) até o Amazon Dash Cart. Essas soluções utilizam um arsenal de tecnologias, incluindo visão computacional avançada e múltiplos sensores para identificar produtos sem a necessidade de escanear códigos [Wankhede et al. 2022]. No entanto, como discutido na Introdução, a complexidade e o custo associados a essa abordagem são proibitivos para a maioria dos varejistas. Em contraste, os sistemas de autoatendimento com balanças representam um passo intermediário, mas cuja implementação frequentemente peca na usabilidade. Nosso projeto se posiciona como uma alternativa pragmática, que aprende com as falhas das soluções existentes para oferecer uma melhoria tangível e acessível, focando na confiabilidade do código de barras e na otimização do fluxo de pagamento.

3. Metodologia de Desenvolvimento e Avaliação

Adotaremos uma metodologia ágil, com o desenvolvimento dividido em três Sprints, culminando na entrega final do protótipo e da documentação.

3.1. Cronograma de Atividades

Sprint	Período	Atividades Principais	Entrega Associada
1	Até 12/09	Definição do tema, pesquisa aprofundada, planejamento. Elaboração e entrega do documento técnico inicial.	Documento Técnico e Apresentação da Sprint #1
–	13/09 – 26/09	Aquisição de hardware (ESP32, células de carga e demais componentes).	–
–	27/09 – 16/10	Implementação do backend, adiantando o software que independe do hardware.	ocumento Técnico e Apresentação da Sprint #2
2	Até 17/10	Consolidação dos resultados preliminares, e coleta de feedbacks da sprint 2.	–
–	18/10 – 07/11	Integração: Conexão entre o ESP32 e a interface do usuário. Montagem do protótipo físico.	–
–	08/11 – 27/11	Refinamento e Testes: Testes de usabilidade, correção de bugs, avaliação de desempenho.	–
3	Até 28/11	Consolidação dos resultados finais. Produção do artigo científico, apresentação final e pitch comercial.	Entrega Final Completa – Documento Técnico, Apresentação da Sprint #3 e Pitch Comercial

Tabela 1. Planejamento das Sprints

3.2. Método de Avaliação/Validação

O método de avaliação adotado será a medição em um protótipo funcional. Após a fase de integração, realizaremos testes controlados para avaliar a eficácia e a usabilidade do sistema. As métricas a serem consideradas incluem:

- **Acurácia e Velocidade de Leitura:** Taxa de sucesso e tempo de resposta na leitura de códigos de barras em diferentes condições de iluminação, distância e ângulo.
- **Confiabilidade do Sistema:** Comparação entre os itens registrados virtualmente e os itens físicos no carrinho ao final de uma simulação de compra.
- **Fluxo de Checkout Expresso:** Simulação do processo de pré-pagamento e verificação final, medindo o tempo economizado em comparação a um checkout tradicional.
- **Usabilidade:** Avaliação qualitativa da interface (questionários, observação), considerando facilidade para adicionar/remover itens e alterar quantidades.

4. Definição do Tema

O objeto de estudo é a viabilidade de um sistema de carrinho de compras inteligente de baixo custo como ferramenta de auxílio ao cliente. O problema está delimitado à otimização da experiência em loja, através da transparência de gastos e da agilização do processo de pagamento.

4.1. Fluxo Operacional do Sistema AutoCart

O projeto AutoCart opera através da interação contínua entre três componentes principais: a Aplicação do Cliente (Frontend), o Hardware Embarcado no Carrinho (Firmware) e o Servidor Central (Backend). O fluxo a seguir descreve a jornada completa, desde a perspectiva do usuário até a lógica interna dos bancos de dados.

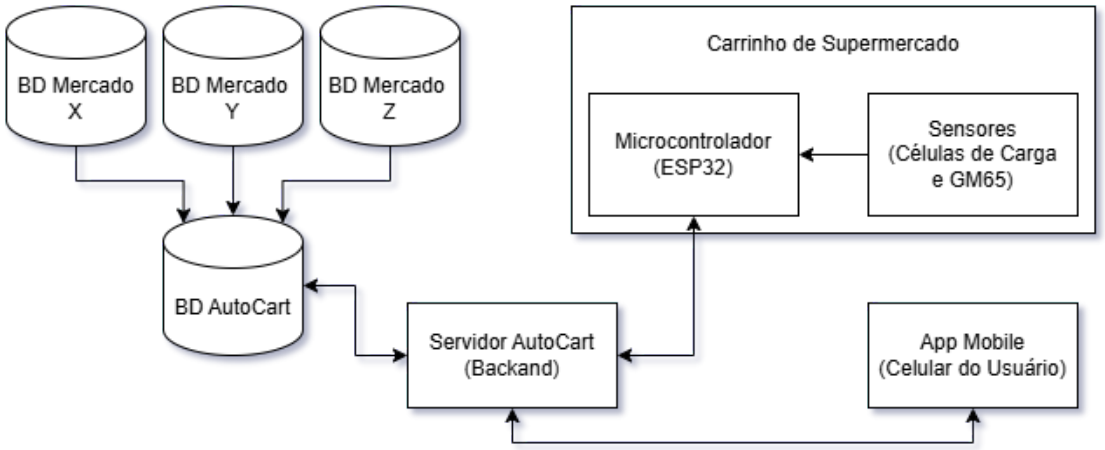

Figura 1. Diagrama da Arquitetura e Fluxo de Dados do Sistema AutoCart.

4.1.1. Preparação e Sincronização (Back-Office)

Antes de qualquer interação do cliente, o sistema depende de uma arquitetura de dados robusta que simula um ambiente de múltiplos mercados:

- **Bancos de Dados dos Mercados:** Cada mercado parceiro é representado por um banco de dados independente (ex: `mercado_a.sqlite`). Este banco contém a *fonte da verdade* para o catálogo de produtos daquela loja específica, incluindo nome, preço, peso e código de barras. Os administradores de cada mercado gerenciam este catálogo através de endpoints de API seguros.
- **Banco de Dados Central:** O sistema AutoCart possui um banco de dados central (`autocart_db.sqlite`) que consolida as informações. Um script de sincronização (`syncProducts.js`) é executado periodicamente para ler os produtos de todos os bancos de mercado e replicá-los no banco central. Cada produto copiado é *etiquetado* com um `mercadoId` para garantir que o sistema saiba a qual loja ele pertence.

Essa arquitetura ETL (Extract, Transform, Load) garante que a API principal, utilizada pelo cliente, seja rápida e resiliente, pois ela consulta apenas o banco central, sem depender da disponibilidade dos sistemas de cada mercado durante a compra.

4.1.2. A Jornada do Cliente

O fluxo do usuário é projetado para ser intuitivo e direto, dividido em quatro etapas principais:

1. **Etapa 1: Autenticação e Início da Compra**
   (a) **Cadastro e Login:** O cliente utiliza o aplicativo mobile para se cadastrar e fazer login. O backend valida as credenciais e retorna um token de autenticação (JWT) que será usado para validar todas as ações futuras.
   (b) **Vinculação com o Carrinho:** Ao entrar na loja, o cliente usa a câmera do aplicativo para escanear um QR Code único presente em um carrinho disponível. O app envia o ID do carrinho para a API.
   (c) **Início da Sessão:** O backend recebe a requisição, verifica se o carrinho está com o status "livre" e cria um novo registro de "Compra" com o status "ativa", associando o `usuarioId` (do token) ao `carrinhoId`. Em seguida, o status do carrinho é atualizado para "em_uso", impedindo que outro cliente o utilize.
2. **Etapa 2: Adicionando Produtos**
   (a) **Escaneamento do Produto:** O cliente escaneia o código de barras de um produto com a câmera do aplicativo.
   (b) **Requisição à API:** O app envia o código de barras para o backend.
   (c) **Validação e Lógica de Negócio:** O backend identifica a compra ativa do cliente e o mercado associado ao carrinho. Ele procura o produto no banco de dados central, cruzando o `codigo_barras` com o `mercadoId` para garantir que o preço e as informações sejam os corretos para aquela loja. O produto é colocado fisicamante dentro do carrinhos de compras, e o seu peso é medido pelas células de carga, se o peso corresponder ao descrito no banco de dados (com a adição da margem de erro, tendo em vista apossibilidade de pequenas variações durante a fabricação), o produto é então adicionado à lista de `ItemCompra`. Se o item já existe, sua quantidade é incrementada e o valor total da compra é recalculado.
   (d) **Feedback Visual:** A API retorna uma confirmação de sucesso, e o aplicativo atualiza a interface, exibindo o novo item na lista e o valor total recalculado.

   OBS.: O carrinho de compras possui um leitor de código de barras GM65 e uma tela touch acoplados em sua estrutura. Caso o cliente não tenha afinidade com o uso do celular, não possua um dispositivo ou simplesmente não queira utilizá-lo, ele poderá recorrer ao hardware embarcado para realizar as mesmas operações descritas na Etapa 2. Dessa forma, busca-se garantir acessibilidade a idosos e pessoas com maior dificuldade no uso de smartphones. Assim, essa interação alternativa, segue a mesma estrutura de operações descritas anteriormente nessa etapa.
3. **Etapa 3: Gerenciamento em Tempo Real**
   Durante toda a compra, o cliente pode usar o aplicativo para:
   - **Visualizar o Carrinho:** A qualquer momento, o app pode solicitar à API o estado atual da compra (`GET /compras/ativa`), que retorna a lista completa de itens, quantidades e o valor total.

- **Remover Itens:** O cliente pode remover um item da lista através do app. O backend recebe a requisição, atualiza a quantidade ou remove o `ItemCompra`, e recalcula o valor total.

4. **Etapa 4: Finalização da Compra**
   (a) **Ação do Cliente:** Ao terminar as compras, o cliente pressiona "Finalizar Compra" no aplicativo.
   (b) **Requisição à API:** O app envia uma requisição para o endpoint de finalização.
   (c) **Encerramento da Sessão:** O backend altera o status da "Compra" para "finalizada" e, crucialmente, atualiza o status do "Carrinho" de volta para "livre", disponibilizando-o para o próximo cliente.
   (d) **Checkout Expresso:** Com a compra finalizada no sistema, o cliente se dirige a um caixa expresso. O pagamento pode ser concluído rapidamente, pois a lista de itens e o valor total já estão processados, eliminando a necessidade de escanear todos os produtos novamente.

Este fluxo integrado demonstra como o hardware (carrinho com QR Code), o software do cliente (app) e a inteligência do servidor (backend e bancos de dados) trabalham em conjunto para criar uma experiência de compra mais eficiente, transparente e moderna.

4.2. Limitações do Trabalho

Para manter o foco no núcleo da proposta e garantir a viabilidade do protótipo dentro do cronograma semestral, as seguintes limitações foram definidas e refinadas ao longo do desenvolvimento:

- **Hardware de Prototipagem:** A solução utiliza componentes de fácil acesso e baixo custo (ESP32, leitor de código de barras GM65, smartphone para simulação de tela). Estes componentes são adequados para a validação do conceito, mas não representam a robustez ou o desempenho de uma solução comercial final.
- **Identificação de Produtos:** O método de identificação de produtos foi refinado para o uso exclusivo de um leitor de código de barras dedicado, em detrimento da proposta inicial com webcam. Consequentemente, produtos sem código de barras, como itens de hortifruti vendidos a granel, permanecem fora do escopo deste trabalho, por enquanto.
- **Verificação de Peso Dependente do Firmware:** A validação de peso dos produtos, uma funcionalidade chave para prevenção de fraudes, não é atualmente imposta pela API do backend. Na fase atual, a lógica de negócio confia na ação de escaneamento realizada pelo usuário, e a responsabilidade pela verificação de peso será delegada inteiramente ao firmware do hardware, representando uma limitação do sistema enquanto o hardware não está integrado.
- **Sistema de Pagamento Conceitual:** O fluxo de checkout expresso é puramente conceitual e foca na otimização do tempo. Nenhuma funcionalidade de transação financeira real foi implementada no escopo deste projeto.
- **Arquitetura de Sincronização Simulada:** A arquitetura multi-tenant, embora funcional para o protótipo, é uma simulação. O uso de bancos de dados SQLite distintos com um script de sincronização manual é uma abordagem de prototipagem e não representa a performance ou a atomicidade de um sistema de produção, que utilizaria um único banco de dados mais robusto com estratégias de isolamento de dados em tempo real.

Os testes serão realizados em cenários controlados (por exemplo: iluminação baixa/média/alta; ângulos variados; distâncias típicas de leitura) com múltiplas repetições por condição e registro de logs para análise. Serão coletadas estatísticas descritivas (média, desvio-padrão) e apresentados gráficos para cada métrica.

5. Desenvolvimento e Resultados Preliminares

Conforme o cronograma da Sprint 2, esta etapa do projeto concentrou-se na construção da infraestrutura de software que servirá de alicerce para os componentes de frontend e hardware. Diante de um atraso na entrega dos componentes eletrônicos, adotou-se uma estratégia de desenvolvimento desacoplado (*software-first*), que permitiu a implementação e validação completa da API (Application Programming Interface) do sistema.

5.1. Desenvolvimento e Implementação

A implementação do backend foi realizada utilizando `Node.js` em conjunto com o framework `Express.js`, uma escolha que favorece a agilidade no desenvolvimento de APIs RESTful. Para a persistência de dados, optou-se pelo ORM (Object-Relational Mapper) Sequelize, abstraindo a complexidade do banco de dados relacional SQLite.

Uma decisão arquitetural chave foi a simulação de um ambiente *multi-tenant*, visando replicar um cenário de negócio realista onde múltiplos mercados poderiam utilizar o sistema. Para isso, foram estruturados bancos de dados SQLite independentes para cada mercado simulado ("Mercado A" e "Mercado B"), contendo seus catálogos de produtos. Um banco de dados central, `autocart_db`, foi criado para gerenciar os dados da aplicação principal, como usuários, carrinhos, compras e uma cópia sincronizada dos produtos. Foi desenvolvido um script de sincronização (`syncProducts.js`) que lê os dados dos bancos dos mercados e os insere ou atualiza no banco central, garantindo a consistência dos dados sem acoplar a API principal aos sistemas externos.

Para a segurança, implementou-se um sistema de autenticação baseado em JSON Web Tokens (JWT), com as senhas dos usuários sendo criptografadas através do algoritmo bcrypt. O controle de acesso foi segregado em dois níveis de permissão (*roles*): `cliente`, para os consumidores finais, e `admin`, para os gestores de mercado. A API foi dividida em três módulos principais de rotas:

- **Autenticação (`/auth`):** Endpoints para cadastro e login de usuários.
- **Administração (`/admin`):** Endpoints protegidos para o CRUD (Create, Read, Update, Delete) de produtos e carrinhos, com a lógica de negócio garantindo que um administrador só possa gerenciar os recursos associados ao seu próprio mercado.
- **Compras (`/compras`):** Endpoints para o fluxo de compra do cliente, incluindo iniciar uma sessão, adicionar e remover itens, visualizar o carrinho em tempo real e finalizar a compra.

5.2. Resultados Preliminares

Ao final desta sprint, o principal resultado obtido é um backend 100% funcional, robusto e seguro. A API RESTful completa foi testada e validada através de ferramentas como o Postman, confirmando que todas as regras de negócio foram implementadas corretamente.

O sistema é capaz de:

- Gerenciar o ciclo de vida completo de uma compra do cliente, desde a autenticação até a finalização.
- Garantir a segurança e o isolamento dos dados entre diferentes mercados.
- Oferecer uma base estável e bem definida (um "contrato") para que as equipes de frontend (aplicativo mobile) e firmware (hardware embarcado) possam iniciar seus desenvolvimentos em paralelo, consumindo os endpoints já existentes.

A arquitetura de sincronização provou-se eficaz, permitindo que a API central opere com alta performance e resiliência, sem depender diretamente da disponibilidade dos sistemas simulados de cada mercado.

5.3. Dificuldades e Soluções

O desenvolvimento nesta etapa enfrentou três desafios principais:

- **Atraso na Entrega do Hardware:** A principal dificuldade foi um fator externo: os componentes eletrônicos (NodeMCU, células de carga, leitor) não chegaram a tempo. Isso impossibilitou a montagem e os testes físicos do protótipo, que eram atividades planejadas para esta fase. A **solução** foi a adoção da estratégia de desenvolvimento desacoplado, priorizando a finalização do backend. Isso mitigou o impacto do atraso e, na prática, reduziu os riscos do projeto, pois a base de software, que contém a maior complexidade lógica, foi concluída e validada antecipadamente.
- **Definição da Arquitetura de Dados:** Conceber uma estrutura que suportasse múltiplos mercados de forma escalável, como descrito no objetivo do projeto de ser uma solução de baixo custo para o varejo, foi um desafio conceitual. A **solução** foi a implementação da arquitetura *multi-tenant* simulada. Em vez de uma abordagem simplista, optou-se por um modelo mais complexo, porém mais alinhado a um produto real, o que enriqueceu tecnicamente o projeto.
- **Problemas Conceituais de Fraude:** Conforme levantado na introdução, lidar com cenários complexos como a troca de etiquetas ou variações de peso de produtos é um desafio que transcende a tecnologia. A **solução** adotada foi a aceitação das limitações do projeto e o foco no valor tangível a ser entregue. Em vez de buscar uma solução utópica e à prova de falhas, o desenvolvimento concentrou-se em resolver as dores principais do consumidor: o controle de gastos em tempo real e a agilização do checkout.

6. Repositório do Projeto

O código-fonte, a documentação e outros artefatos do projeto estão disponíveis no seguinte repositório do GitHub:

`https://github.com/DomynicBl/AutoCart`

Referências

Barry, R. (2010). *Using the FreeRTOS Real-Time Kernel*. Real Time Engineers Ltd.

Espressif Systems (2023). ESP32-WROOM-32E & ESP32-WROOM-32UE Datasheet. Technical report, Espressif Systems.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

Kopetz, H. (2011). *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Science & Business Media.

Wankhede, S. A., Dhabu, M. S., Yawle, P. M., and Sherekar, S. S. (2022). Smart shopping cart: a systematic literature review. *Materials Today: Proceedings*, 62:4363–4369.