CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

RAFAEL REIS COSTA

**SISTEMA GERENCIADOR DE GRÃOS PARA DESPENSA DE ALIMENTOS FAMILIAR**

RIO DO SUL

2024

CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

RAFAEL REIS COSTA

**SISTEMA GERENCIADOR DE GRÃOS PARA DESPENSA DE ALIMENTOS FAMILIAR**

Trabalho de Conclusão de Curso a ser apresentado ao curso de Sistemas da Informação, da Área das Ciências Naturais, da Computação e das Engenharias, do Centro Universitário para o Desenvolvimento do Alto Vale do Itajaí, como condição parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Prof. Orientador: Marciel Linz

RIO DO SUL

2024

CENTRO UNIVERSITÁRIO PARA O DESENVOLVIMENTO DO ALTO VALE DO ITAJAÍ - UNIDAVI

rafael reis costa

**SISTEMA GERENCIADOR DE GRÃOS PARA DESPENSA DE ALIMENTOS FAMILIAR**

Trabalho de Conclusão de Curso a ser apresentado ao curso de Sistemas da Informação, da Área das Ciências Naturais, da Computação e das Engenharias, do Centro Universitário para o Desenvolvimento do Alto Vale do Itajaí- UNIDAVI, a ser apreciado pela Banca Examinadora, formada por:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Professor Orientador: Marciel de Linz

Banca Examinadora:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Prof.

Rio do Sul, 07 de julho de 2024.

Se o PMA puder descrever neste tópico do Twitter exatamente como US$ 6 bilhões resolverão a fome mundial, venderei ações da Tesla agora mesmo e farei isso (Elon Musk).

Para o todo.

AGRADECIMENTOS

A todos que suportaram a falta da minha presença e aqueles que de alguma maneira, direta ou indiretamente, contribuíram para que este trabalho fosse concluído.

RESUMO

A fome é um problema mundial. Com o intuito de criar uma solução que possa mitigar de alguma maneira o desperdício de alimentos foi concebida a ideia de construir um protótipo que ofereça controle e gerenciamento sobre a quantidade, em um primeiro momento, em quilograma e posteriormente volume em litros. O protótipo visa utilizar tecnologias e linguagens de programação consolidadas e maduras como .Net e C# que são amplamente utilizadas no mundo corporativo e as tecnologias IoT contemporâneas e modernas que cada vez mais fazem parte do cotidiano dos seres humanos. O estudo de eletrônica e redes também são partes importantes que compõe este trabalho pois, sem elas não há como integrar software e hardware.

**Palavras-Chave:** Despensa familiar, IoT, Automação.

ABSTRACT

Hunger is a worldwide problem. In order to create a solution that can mitigate food waste in some way, the idea was conceived of building a prototype that offers control and management over quantity, at first in kilograms and later volume in liters. The prototype aims to use consolidated and mature technologies and programming languages such as .Net and C#, which are widely used in the corporate world, and contemporary and modern IoT technologies that are increasingly becoming part of human daily life. The study of electronics and networks are also important parts of this work because, without them, there is no way to integrate software and hardware.

**Keywords:** Family pantry, Automation, IoT.

LISTA DE QUADROS

Quadro 1 - Características gerais do banco de dados PostgreSQL 15

Quadro 2 - Funcionamento do React Native 17

Quadro 3 - Porque C# 19

Quadro 4 - Pontos de Design 20

Quadro 5 - Componentes 20

Quadro 6 – Implementações .NET 21

Quadro 7 – Tipos de API 21

Quadro 8 – Seis restrições REST 22

Quadro 9 – Categorias dos sensores 25

Quadro 10 – Categorias dos atuadores 26

LISTA DE ABREVIATURAS E SIGLAS

XML Extensible Markup Language

JSX JavaScript Syntax Extension

JS JavaScript

HTML HyperText Markup Language

CSS Cascading Style Sheets

DOM Document Object Model

IoT Internet of Things – Internet das Coisas

SoCs System on a Chip – Sistema no Chip

**SUMÁRIO**

1. INTRODUÇÃO 12

1.1. PROBLEMA DE PESQUISA 13

1.2. OBJETIVOS 13

1.2.1. Geral 13

1.2.2. Específicos 13

1.3. JUSTIFICATIVA 13

2. REFERENCIAL TEÓRICO 15

2.1. BANCO DE DADOS 15

2.1.1. PostgresSQL - Curta história e definição 15

2.1.2. Conceito 15

2.1.3. Características 16

2.1.4. Porque PostgreSQL? 16

2.2. APLICAÇÃO MOBILE 16

2.2.1. React Native 17

2.3. BACKEND 19

2.3.1. C# 19

2.3.2. Porque C# 20

2.3.3. .Net framework 20

2.3.4. Api 22

2.3.4.1. Tipos de API 22

2.3.4.2. Padrão REST 23

2.4. IOT – INTERNET DAS COISAS 24

2.4.1. Conectividade 25

2.4.2. Ecossistema 26

2.4.2.1. Sensores 26

2.4.2.2. Atuadores 27

2.4.2.3. Comunicações 27

1. introdução

A fome é um problema que atinge milhares de pessoas ao redor do mundo. A população dos países mais pobres são os que sofrem mais. Com base no relatório SOFI 2024, estima-se que entre 713 e 757 milhões de pessoas, o que corresponde a 8,9 e 9,4 por cento da população mundial, respetivamente, poderão ter enfrentado a fome em 2023. Isto é igual a mais que a população dos Estados Unidos e Brasil juntos.

Em contrapartida o Relatório do Índice de Desperdício de Alimentos 2024, aponta que em 2022, houve um desperdício mundial aproximado de 1,05 bilhão de toneladas de alimento, que significa que um quinto dos alimentos disponíveis para as pessoas foram jogados fora por famílias, restaurantes e setor alimentício. Há desperdício de alimentos tangivelmente considerável em várias partes do mundo.

A disparidade entre a fome e o desperdício de alimentos aponta para uma necessidade urgente de soluções inovadoras que possam otimizar o uso dos recursos alimentares e garantir que mais pessoas tenham acesso à alimentação adequada. Neste contexto, a tecnologia emergente da Internet das Coisas (IoT) oferece promissoras possibilidades de intervenção. O IoT permite a interconexão de dispositivos e sistemas através da internet, facilitando a coleta, análise e utilização de dados em tempo real.

Este trabalho de conclusão de curso propõe o desenvolvimento de uma aplicação integrada com um sistema IoT, destinada a reduzir o desperdício de alimentos. A solução tecnológica consiste em um sistema inteligente que monitora e gerencia os estoques de alimentos em diferentes pontos da cadeia de suprimentos. Através do uso de sensores e algoritmos de aprendizado de máquina, o sistema poderá prever a deterioração dos alimentos, otimizar os níveis de estoque e sugerir ações corretivas para minimizar perdas. Além disso, o sistema pode integrar-se às plataformas de distribuição para redirecionar alimentos próximos da data de vencimento para organizações que lidam com a insegurança alimentar, contribuindo para um uso mais eficiente e sustentável dos recursos alimentares.

Assim, esta aplicação não apenas almeja reduzir o desperdício de alimentos, mas também busca criar um impacto positivo no combate à fome, promovendo uma distribuição mais equitativa e consciente dos alimentos disponíveis.

1.1 PROBLEMA DE PESQUISA

Como efetuar controle de estoque da despensa familiar de alimentos de forma automatizada?

1.2 Objetivos

Neste capítulo serão abordados os objetivos gerais e específicos.

1.2.1 Geral

* Desenvolver um protótipo para pesagem de alimentos com uma aplicação para controlar e gerenciar a quantidade dos itens em uma despensa.

1.2.2 Específicos

* Levantar requisitos funcionais e não funcionais para o desenvolvimento do software.
* Criar o gerenciamento do projeto no Microsoft Project.
* Desenhar o protótipo e esquemas elétricos, definindo o hardware que será utilizado.
* Efetuar a modelagem do Banco de Dados.
* Construir a aplicação mobile.
* Integrar com o sistema, realizando testes.

1.3 Justificativa

Em um mundo onde milhões de pessoas ainda sofrem de fome enquanto bilhões de toneladas de alimentos são desperdiçadas, a aplicação que está sendo desenvolvida não é apenas uma solução; é um avanço revolucionário na forma como gerenciamos nossos recursos alimentares.

Imagine um sistema que combina o poder da tecnologia IoT com inteligência artificial para monitorar e gerenciar em tempo real os estoques de alimentos, diretamente na despensa de cada casa ou estabelecimento. Esse protótipo de pesagem de alimentos, integrado a uma aplicação sofisticada, tem o potencial de transformar radicalmente a maneira como lidamos com os alimentos, prevenindo excessos e maximizando a eficiência.

Não se trata apenas de reduzir o desperdício – embora essa seja uma vitória monumental por si só, dado que 1,05 bilhão de toneladas de alimentos foram desperdiçadas globalmente em 2022. Este sistema vai além, antecipando necessidades, sugerindo compras de maneira otimizada e até redirecionando alimentos que estão próximos da data de vencimento para onde eles são mais necessários, como bancos de alimentos e organizações de caridade.

Essa solução coloca o controle da sustentabilidade alimentar nas mãos dos usuários, permitindo que cada pessoa, família ou empresa se torne parte ativa na luta contra a fome global. Com essa aplicação, não estamos apenas economizando alimentos; estamos salvando vidas, promovendo uma economia circular e criando um futuro em que, a segurança alimentar, seja uma realidade para todos.

Em termos de inovação e impacto social, essa aplicação representa o ápice do que a tecnologia moderna pode alcançar. Ela não apenas responde a uma necessidade urgente, mas o faz de maneira que transforma profundamente a forma como o mundo consome e distribui alimentos. É, sem dúvida, a solução que o mundo estava esperando para enfrentar dois dos maiores desafios da atualidade: o desperdício de alimentos e a fome.

Em consideração às diversas partes do mundo que sofrem com a falta de alimentos e às pessoas que são diretamente impactadas, é mais do que justificável o desenvolvimento de um meio ou uma ferramenta que ajude a economizar e gerir este importante recurso que deveria ser acessível a todos.

* Implementar um controle automatizado para controle de grãos e outros alimentos para uma despensa de doméstica.
* Utilizar balança ou dispositivo adaptado para realizar a medição da quantidade, em quilograma, dos itens estocados.
* Implementar uma aplicação mobile para inserir, cadastrar e gerenciar os itens da despensa.
* Ser acessível de qualquer lugar.
* Com base na quantidade de itens disponíveis, sugerir receitas.

2. REFERENCIAL TEÓRICO

O conteúdo deste referencial teórico abordará assuntos relacionas às ferramentas e métodos que serão utilizadas para o desenvolvimento do protótipo e software.

2.1. BANCO DE DADOS

Neste capítulo será abordado o banco de dados PostgreSQL, uma breve história, definições e conceitos.

2.1.1. PostgresSQL - Curta história e definição

O PostgresSQL herdou seu nome do banco de dados relacional, Ingres, desenvolvido pelo Professor Michael Stonebraker. Em 1986, o Professor Stonebraker iniciou um projeto chamado POSTGRES, traduzindo do inglês “Pós-Ingres”.

PostgreSQL recebeu seu nome de seu ancestral: Ingres. O Ingres era um banco de dados relacional desenvolvido pelo Professor Michael Stonebraker. Em 1986, o Professor Stonebraker iniciou um projeto Pós-Ingres (do inglês Post Ingres) para desenvolver novos recursos interessantes no cenário de banco de dados e deu a esse projeto o nome de projeto POSTGRES (POST-Ingres). (Ferrari; Pirozzi, 2020)

O PostgresSQL é um sistema gerenciador de base de dados de objetos relacionais (SGBDOR), baseado no POSTGRES, Versão 4.21, desenvolvido na Universidade da Califórnia em Berkeley pelo Departamento de Ciência da Computação. O POSTGRES foi pioneiro em muitos conceitos que somente se tornaram disponíveis em alguns sistemas de base de dados comerciais muito tempo depois. (Ferrari; Pirozzi, 2020).

2.1.2. Conceito

Segundo a documentação oficial, o PostgreSQL é um sistema de gerenciamento de banco de dados relacional (SGDB).” (PostgreSQL, 2024).

Em outras palavras, é um sistema que gerencia dados armazenados através de relações que em síntese é um termo matemático para tabela. Isso significa que ele é um sistema para gerenciar dados armazenados em relações. Relação é essencialmente um termo matemático para tabela. A noção de armazenar dados em tabelas é tão comum hoje em dia que pode parecer inerentemente óbvia, mas há várias outras maneiras de organizar os dados em um banco de dados relacional. Arquivos e diretórios em sistemas operacionais do tipo Unix são um exemplo de banco de dados hierárquico. Um desenvolvimento mais moderno é o banco de dados orientado a objetos. (PostgreSQL, 2024)

2.1.3. Características

O Quadro 1 ilustra algumas características gerais sobre o PostgreSQL.

**Quadro 1 – Características gerais do banco de dados PostgreSQL**

|  |  |
| --- | --- |
| **Característica** | **Descrição** |
| Tamanho máximo | Ilimitado |
| Máximo de uma tabela | 32 Tera Bytes |
| Máximo de linha | 1.6 Tera Bytes |
| Máximo de um campo | 1 Giga Bytes |
| Máx. de linhas por tabela | Ilimitado |
| Máx. de colunas por tabela | 250 - 1600 |
| Máx. de índices por tabela | Ilimitado |

Fonte: Elaborado a partir de (Carvalho, 2017)

**2.1.4 Porque PostgreSQL?**

O PostgreSQL vem sendo amplamente utilizado nas diversas áreas de pesquisa e produção.

O POSTGRES tem sido usado para implementar muitos aplicativos diferentes de pesquisa e produção. Entre eles estão: um sistema de análise de dados financeiros, um pacote de monitoramento de desempenho de motores a jato, um banco de dados de rastreamento de asteroides, um banco de dados de informações médicas e vários sistemas de informações geográficas. (PostgreSQL,2024)

Por consequência amplamente utilizado em universidades e para fins educacionais. Também foi usado como uma ferramenta educacional em várias universidades. Finalmente, a Illustra Information Technologies (mais tarde incorporada à Informix, que agora pertence à IBM) pegou o código e o comercializou. No final de 1992, o POSTGRES tornou-se o principal gerenciador de dados do projeto de computação científica Sequoia 2000. (PostgreSQL 2024)

2.2 APLICAÇÃO MOBILE

Neste capítulo serão abordadas as ferramentas e linguagens utilizadas para o desenvolvimento do aplicativo mobile.

**2.2.1 React Native**

React Native é uma estrutura para criar aplicativos móveis nativos em JavaScript usando a biblioteca React JavaScript; o código do React Native é compilado em componentes nativos reais. Permite que o frontend de uma aplicação seja exibido de forma nativa ao contrário de outros frameworks de aplicativos híbridos que empacotam as visualizações Web criadas com HTML e CSS. (Dabit, 2019)

Com o React Native, os desenvolvedores podem criar exibições nativas e acessar componentes específicos da plataforma nativa usando JavaScript. Isso diferencia o React Native de outros frameworks de aplicativos híbridos, como o Cordova e o Ionic, que empacotam visualizações da Web criadas com HTML e CSS em um aplicativo nativo. Em vez disso, o React Native usa JavaScript e o compila em um verdadeiro aplicativo nativo que pode usar APIs e componentes específicos da plataforma. (Dabit, 2019)

Desempenho e velocidade se mostram melhores do que aplicações criadas em estruturas híbridas além da reutilização de código.

Há muitos benefícios em escolher o React Native como uma estrutura de aplicativo móvel. Como o aplicativo renderiza componentes e APIs nativos diretamente, a velocidade e o desempenho são muito melhores do que com estruturas híbridas, como Cordova e Ionic. Com o React Native, estamos escrevendo aplicativos inteiros usando uma única linguagem de programação: JavaScript. Podemos reutilizar muito código, reduzindo assim o tempo necessário para enviar uma aplicação multiplataforma. (Dabit, 2019)

O React Native utiliza uma mistura de JavaScript e marcação do tipo XML, o JSX.

Semelhante ao React para a Web, os aplicativos React Native são escritos usando uma mistura de JavaScript e marcação do tipo XML, conhecida como JSX. Em seguida, sob o capô, a “ponte” do React Native invoca as APIs de renderização nativas em Objective-C (para iOS) ou Java (para Android). Assim, seu aplicativo será renderizado usando componentes reais de UI móvel, e não webviews, e terá a mesma aparência e comportamento de qualquer outro aplicativo móvel. O React Native também expõe interfaces JavaScript para APIs de plataforma, para que seus aplicativos React Native possam acessar recursos da plataforma, como a câmera do telefone ou a localização do usuário. (Eisenman, 2016)

Para compreender como o React Native trabalha, precisamos analisar o funcionamento do React que manipula o DOM calculando as alterações necessárias em memória e renderiza a quantidade mínima ao invés de renderizar diretamente as alterações na página como cita Eisenman. No React, o Virtual DOM atua como uma camada entre a descrição do desenvolvedor de como as coisas devem parecer e o trabalho feito para realmente renderizar seu aplicativo na página. Para renderizar interfaces de usuário interativas em um navegador, os desenvolvedores devem editar o DOM, ou Modelo de Objeto de Documento. Essa é uma etapa custosa, e gravações excessivas no DOM têm um impacto significativo no desempenho. Em vez de renderizar diretamente as alterações na página, o React calcula as alterações necessárias usando uma versão em memória do DOM e renderiza a quantidade mínima necessária. (Eisenman, 2016)

No Quadro 2 é possível visualizar algumas informações sobre como o framework React Native opera.

**Quadro 2 –** **Funcionamento do React Native**

|  |  |
| --- | --- |
| JSX | Tanto o React quanto o React Native incentivam o uso do JSX. O JSX é basicamente uma extensão de sintaxe do JavaScript que se assemelha ao XML. Você pode criar componentes React Native sem JSX, mas o JSX torna o React e o React Native muito mais legíveis e fáceis de manter. O JSX pode parecer estranho no início, mas é extremamente poderoso, e a maioria das pessoas acaba gostando dele. |
| Threading | Todas as operações de JavaScript, ao interagir com a plataforma nativa, são feitas em um thread separado, permitindo que a interface do usuário e todas as animações sejam executadas sem problemas. Esse thread é onde o aplicativo React reside e onde todas as chamadas de API, eventos de toque e interações são processados. Quando há uma alteração em um componente com suporte nativo, as atualizações são agrupadas e enviadas para o lado nativo. Isso acontece no final de cada iteração do loop de eventos. Para a maioria dos aplicativos React Native, a lógica de negócios é executada no thread JavaScript. |
| React | Um ótimo recurso do React Native é que ele usa o React. O React é uma biblioteca JavaScript de código aberto que também é apoiada pelo Facebook. Ela foi originalmente projetada para criar aplicativos e resolver problemas na Web. Essa estrutura se tornou extremamente popular desde seu lançamento, com seu lançamento, com muitas empresas estabelecidas tirando proveito de sua renderização rápida, facilidade de manutenção e interface de usuário declarativa, entre outras coisas.  A manipulação tradicional do DOM é lenta e cara em termos de desempenho e deve ser minimizada. O React contorna o DOM tradicional com algo chamado DOM virtual: basicamente, uma cópia do DOM real na memória que só muda quando comparando novas versões do DOM virtual com versões antigas do DOM virtual. Isso minimiza o número de operações DOM necessárias para alcançar o novo estado. |
| Fluxo de dados unidirecional | O React e o React Native enfatizam o fluxo de dados unidirecional ou único sentido. Devido à como os aplicativos React Native são criados, esse fluxo de dados unidirecional é fácil de obter. |
| Diffing (diferenciação) | O React usa a ideia de diferenciação e a aplica a componentes nativos. Ele pega sua interface do usuário e envia a menor quantidade de dados para o thread principal para renderizá-la com componentes nativos. A interface do usuário é renderizada de forma declarativa com base no estado, e o React usa o *diffing* para enviar as alterações necessárias pela ponte. |
| Componentes | Ao criar uma interface de usuário no React Native, é útil pensar em seu aplicativo como sendo composto por uma coleção de componentes. Pensando em como uma página é configurada, você já faz isso conceitualmente, mas usando conceitos, nomes ou nomes de classe como header, footer, body, sidebar e assim por diante. Com o React Native, você pode dar a esses componentes nomes que façam sentido para você e para outros desenvolvedores que possam estar usando seu código, facilitando a entrada de novas pessoas em um projeto ou a transferência de um projeto para outra pessoa. |

Fonte: Elaborado a partir de (Eisenman, 2016)

Dessa forma um dos objetivos de construir uma aplicação em React Native é produzir com mínimo de código e lógica específico da plataforma.

Um dos objetivos finais do projeto React Native é ter uma quantidade mínima de lógica e código específicos da plataforma. A maioria das APIs pode ser criada de modo que o código específico da plataforma seja abstraído pela estrutura, oferecendo deste modo, uma única maneira de interagir com elas e criar facilmente uma funcionalidade entre plataformas. (Eisenman, 2016)

2.3 BACKEND

Neste capítulo serão explorados os temas referentes ao desenvolvimento backend da aplicação como ambiente de desenvolvimento, linguagem e framework.

2.3.1 C#

Segundo Ledur (2018) o C# é uma linguagem de programação desenvolvida pela Microsoft dentro da iniciativa .NET. e hoje, é amplamente utilizada em todo o mundo para a criação de sistemas robustos.

Um dos aspectos importantes no desenvolvimento de qualquer aplicação é a segurança, sendo assim, uma das características da linguagem C# é a tipagem estática que impede que haja incompatibilidade entre os dados das variáveis. Que uma variável que deveria receber um valor de tipo inteiro por exemplo, receba um valor do tipo booleano ou string, e isso quer dizer que os valores são testados também em tempo de execução.

O C# suporta tipagem estática, o que significa que a linguagem impõe a segurança de tipos em tempo de compilação. Isso é um acréscimo à segurança de tipo que é imposta em tempo de execução. (Albahari, 2021).

A orientação a objetos é fundamental o desenvolvimento de qualquer aplicação e regra de negócios, assim o C# permite que o código seja fácil de ser escritos e de fácil manutenção como cita o autor, a orientação a objetos permitiu um grande avanço no desenvolvimento de sistemas em todo o mundo. Nesse sentido, ela possibilitou a criação de um código de fácil manutenção e com abstrações importantes para a produtividade. (Ledur, 2018)

2.3.2. Porque C#

A linguagem C# é neutra em termos de plataforma e funciona com uma gama de tempos de execução específicos da plataforma. (Albahari, 2021)

Por ser uma linguagem robusta, segura, altamente tipada e orientada a objetos foi escolhida como linguagem para a construção da API REST para este TCC, vinculando assim todas as características e atributos necessários para a programação da aplicação, como é citado, “C# é uma linguagem de programação multiparadigmática de tipagem forte, imperativa, declarativa, funcional, genérica, orientada a objetos (baseada em classes) e orientada a componentes.”. (Ledur, 2018)

Um dos pontos que posso citar para a escolha da linguagem C# é em relação à experiência do desenvolvedor, não sendo isto um atributo humano, mas o que ela proporciona, oferecendo o suporte necessário para escrever o código de maneira limpa.

A diferença está na experiência do desenvolvedor. Algumas linguagens são realmente boas em orientá-lo a escrever código limpo, enquanto outras não são. O C# não é perfeito, mas tenta ajudá-lo nesse aspecto.”. (Rodenburg, 2021)

**Quadro 3 – Porque C#**

|  |  |
| --- | --- |
| C# é econômico | O C# é gratuito para uso e desenvolvimento.  A linguagem e a plataforma são totalmente de código aberto.  Não é necessária nenhuma taxa de licença para o tempo de execução, e você pode implantar o produto final onde quiser. |
| C# é passível de manutenção | Capacidade de manutenção, capacidade de corrigir bugs, alterar funcionalidades e resolver outros problemas sem efeitos colaterais indesejados.  Possui recursos que melhoram a capacidade de manutenção (e, portanto, a extensibilidade segura) de grandes bases de código. Pense, por exemplo, nos genéricos e no Language-Integrated Query (LINQ).  O desenvolvimento empresarial é a base do desenvolvimento do C# e é onde o C# e o .NET se destacam.  Código auto documentado, significa um código escrito com clareza suficiente para que não sejam necessários comentários para explicar a lógica. O código documenta a si mesmo. Pode ser executado em várias plataformas, embora com algumas limitações (algumas plataformas mais antigas são restritas a versões mais antigas do C# e do .NET Framework). |

Fonte: Elaborado a partir de (RODENBURG, 2021)

**2.3.3. .Net framework**

Segundo a documentação oficial da Microsoft “O .NET é uma plataforma de desenvolvimento gratuita multiplataforma e de código aberto para criar muitos tipos de aplicativos.” (Microsoft, 2024)

A palavra .NET tem sido usada há muito tempo como um termo abrangente para qualquer tecnologia que inclua a palavra .NET (.NET Framework, .NET Core, .NET Standard e assim por diante). (Albahari, 2021)

Contudo, a plataforma .Net da Microsoft e a linguagem de programação C# são fortemente ligadas.

A plataforma .NET da Microsoft e a linguagem de programação C# foram formalmente introduzidas por volta de 2002 e rapidamente se tornaram um pilar do desenvolvimento de software moderno. (Troelsen; Japikse, 2022)

Um dos recursos muito bons que o .NET possui é o coletor de lixo que é responsável pelo gerenciamento de memória, que decide como e onde os dados serão alocados como também quando e de que maneira irá liberar memória.

A plataforma .NET foi projetada para fornecer produtividade, desempenho, segurança e confiabilidade. Ele fornece gerenciamento automático de memória por um coletor de lixo (GC). (Microsoft, 2024)

O Quadro 4 mostra os pontos de design do .NET.

**Quadro 4 –** **Pontos de Design**

|  |  |
| --- | --- |
| Produtividade | Pilha completa com runtime, bibliotecas, idioma e ferramentas, contribuindo para a experiência do usuário do desenvolvedor. |
| Código seguro | Modelo de computação primário, enquanto o código não seguro permite otimizações manuais adicionais. |
| Código estático e dinâmico | Tem suporte, permitindo um amplo conjunto de cenários distintos.  A Interoperabilidade de código nativo e intrínsecos de hardware são de baixo custo e alta fidelidade (API bruta e acesso de instrução). |
| Interoperabilidade de código nativo e intrínsecos de hardware | Baixo custo e alta fidelidade (API bruta e acesso de instrução). |
| Código é portátil entre plataformas | Sistema operacional e arquitetura de chip, enquanto a segmentação por plataforma permite especialização e otimização. |
| Adaptabilidade entre domínios de programação | Nuvem, cliente, jogos, está habilitada com implementações especializadas do modelo de programação de uso geral. |
| Padrões do setor | OpenTelemetry e gRPC, são favorecidos em relação às soluções sob medida. |

Fonte: Elaborado a partir de (Microsoft, 2024)

O .NET opera com diversos componentes. O Quadro 5 traz algumas sobre esses componentes.

**Quadro 5 – Componentes**

|  |  |
| --- | --- |
| Runtime | Executa o código do aplicativo. |
| Bibliotecas | Fornece funcionalidade de utilitário como Análise JSON. |
| Compilador | Compila o código-fonte C# (e outros idiomas) no código executável (runtime) |
| SDK | Habilita a criação e o monitoramento dos aplicativos com fluxos de trabalho modernos. |
| Pilhas de aplicativos | Como ASP.NET Core e Windows Forms, que permitem a gravação dos aplicativos. |

Fonte: Elaborado a partir de (Microsoft, 2024)

A Microsoft informa que dentro do ecossistema do .NET há diversas variantes e ambientes que são parte do histórico e parte técnico. O Quadro 6 cita as implementações do .NET.

**Quadro 6 – Implementações .NET**

|  |  |
| --- | --- |
| .NET Framework | O .NET original. O .NET Framework é a implementação original do .NET que existe desde 2002. Ele fornece acesso aos amplos recursos do Windows e do Windows Server. Ele tem suporte ativo, em manutenção. O .NET Framework é otimizado para a compilação de aplicativos da área de trabalho do Windows. |
| Mono | A comunidade original e o .NET de software livre. Uma implementação multiplataforma do .NET Framework. O Mono é uma implementação do .NET que é usada principalmente quando um pequeno runtime é necessário. É o runtime que executa os aplicativos Xamarin no Android, no macOS, no iOS, no tvOS e no watchOS e ele se concentra principalmente em um volume pequeno.  O Mono normalmente é usado com um compilador Just-In-Time, mas ele também apresenta um compilador estático completo (compilação Ahead Of Time) que é usado em plataformas como iOS. |
| .NET (Core) | .NET moderno. Uma implementação multiplataforma e de código aberto do .NET, repensada para a era da nuvem e que permanece significativamente compatível com o .NET Framework. Suportado ativamente para Linux, macOS e Windows. |

Fonte: Elaborado a partir de (Microsoft, 2024)

**2.3.4. API**

A Application Programming Interface tem sido amplamente utilizada na programação de aplicações e é uma interface de programação de aplicativos (API) é um tipo de interface de software que expõe ferramentas e serviços que os programas de computador usam para interagir uns com os outros e trocar informações. (Sanctis, 2023)

2.3.4.1. Tipos de API

O Quadro 7 mostra os tipos de API’s existentes.

**Quadro 7 –** **Tipos de API**

|  |  |
| --- | --- |
| APIs Publicas | As APIs públicas também são chamadas de APIs abertas. Como o nome sugere, esse termo se refere a APIs que devem estar disponíveis para uso por qualquer terceiro, geralmente sem limitações de acesso. Essas APIs normalmente não envolvem autenticação e autorização (se forem de uso livre) ou empregam um mecanismo de autenticação de chave ou token se precisarem identificar o chamador por vários motivos (como a aplicação de custos por chamada). |
| APIs de parceiros | As APIs que se enquadram nessa categoria estão disponíveis somente para parceiros especificamente selecionados e autorizados, como desenvolvedores externos, empresas integradoras de sistemas, protocolos de Internet (IPs) externos na lista de permissões e similares. As APIs de parceiros são usadas principalmente para facilitar as atividades business-to-business (B2B). Essas APIs normalmente implementam mecanismos de autenticação fortes e técnicas de restrição de IP para evitar que sejam acessadas por usuários não autorizados; elas definitivamente não se destinam a ser consumidas por usuários finais ou clientes "públicos", como sites padrão. |
| APIs internas | Também conhecidas como APIs privadas, essas APIs destinam-se apenas ao uso interno, como para conectar diferentes serviços pertencentes à mesma organização, e geralmente são hospedadas na mesma rede privada virtual, web farm ou nuvem privada. Uma API interna, por exemplo, pode ser usada pelo software interno de planejamento de recursos empresariais para recuperar dados de várias fontes internas de negócios (folha de pagamento, gerenciamento de ativos, gerenciamento de projetos, compras etc.), além de criar relatórios de alto nível e visualizações de dados. |
| APIs compostas | Chamadas muitas vezes de gateways de API, essas APIs combinam várias APIs para executar uma sequência de operações relacionadas ou interdependentes com uma única chamada. Permitem que os desenvolvedores acessem vários pontos de extremidade ao mesmo tempo. Essa abordagem é usada principalmente em padrões de arquitetura de micro serviços, nos quais a execução de uma tarefa complexa pode exigir a conclusão de uma cadeia de subtarefas tratadas por vários serviços de forma síncrona ou assíncrona. Uma API composta atua principalmente como um orquestrador de API, garantindo que as várias subtarefas necessárias para executar a chamada principal sejam bem-sucedidas e capazes de retornar um resultado válido (ou invalidar todo o processo se não forem). |

Fonte: Elaborado a partir de (Sanctis, 2023)

2.3.4.2. Padrão REST

O padrão de arquitetura REST (Representational State Transfer) adotado e utilizado mundialmente que se constitui de um protocolo de boas práticas para a construção de software. Para o desenvolvimento da aplicação será o implementado o padrão arquitetônico REST para trazer maior qualidade e legibilidade do código em geral.

A transferência de estado representativo (REST) é um estilo arquitetônico projetado especificamente para aplicativos baseados em rede que usam os métodos de solicitação HTTP GET, POST, PATCH, PUT e DELETE padrão (definidos pela primeira vez na RFC 2616) para acessar e manipular dados. Seu paradigma arquitetônico se baseia em seis restrições de orientação que, se implementadas corretamente, podem beneficiar muito várias propriedades da API da Web. (Sanctis, 2023)

O Quadro 8 relaciona as seis restrições do padrão REST.

**Quadro 8 – Seis restrições REST**

|  |  |
| --- | --- |
| Abordagem cliente-servidor | as APIs RESTful devem aplicar o princípio da separação de responsabilidades, mantendo separadas as responsabilidades da interface do usuário e o armazenamento de dados. Essa abordagem é especialmente adequada para a World Wide Web, onde os clientes (como navegadores) têm uma função separada dos aplicativos da Web e não sabem nada sobre como eles foram implementados. |
| Ausência de estado | O servidor deve tratar todas as comunicações entre clientes sem manter, armazenar ou reter dados de chamadas anteriores (como informações da sessão). Além disso, o cliente deve incluir todas as informações relacionadas ao contexto, como chaves de autenticação, em cada chamada. Essa abordagem reduz a sobrecarga de cada solicitação no servidor, o que pode melhorar |

**Continuação do Quadro 8 – Seis restrições REST**

|  |  |
| --- | --- |
|  | significativamente as propriedades de desempenho e dimensionamento da API da Web, especialmente sob carga pesada. |
| Capacidade de armazenamento em cache | Os dados trocados entre o cliente e o servidor devem usar os recursos de armazenamento em cache fornecidos pelo protocolo HTTP. Mais especificamente, todas as respostas HTTP devem conter as informações apropriadas de cache (ou não cache) em seus cabeçalhos para otimizar a carga de trabalho do cliente e, ao mesmo tempo, evitar que ele forneça conteúdo obsoleto ou desatualizado por engano. Uma boa estratégia de armazenamento em cache pode ter um grande impacto nas propriedades de escalabilidade e desempenho da maioria das APIs da Web. |
| Sistema em camadas | Colocar o servidor atrás de um ou mais serviços ou filtros HTTP intermediários, como encaminhadores, proxies reversos e balanceadores de carga, pode melhorar muito os aspectos gerais de segurança de uma API da Web, bem como suas propriedades de desempenho e escalabilidade. |
| Código sob demanda (COD – Code On Demand) | COD é a única restrição REST opcional. Ela permite que o servidor forneça código executável ou scripts que os clientes podem usar para adotar um comportamento personalizado. Exemplos de COD incluem computação distribuída e técnicas de avaliação remota, nas quais o servidor delega parte de seu trabalho aos clientes ou precisa que eles executem determinadas verificações localmente usando tarefas complexas ou personalizadas, como verificar se alguns aplicativos ou drivers estão instalados. Em um sentido mais amplo, essa restrição também se refere à capacidade - rara nos primórdios do REST, mas comum na maioria dos aplicativos da Web recentes com tecnologia JavaScript - de buscar no servidor o código-fonte necessário para criar e carregar o aplicativo, bem como para solucionar problemas com outras chamadas REST. O COD pode melhorar o desempenho e o dimensionamento de uma API da Web, mas, ao mesmo tempo, reduz a visibilidade geral e apresenta riscos de segurança não triviais, razão pela qual é sinalizado como opcional. |
| Interface uniforme | A última restrição REST é a mais importante. Ela define os quatro recursos fundamentais que uma interface RESTful deve ter para permitir que os clientes se comuniquem com o servidor sem que o servidor saiba nada sobre como eles funcionam, mantendo-os desacoplados da implementação subjacente da API da Web. |

Fonte: Elaborado a partir de (Sanctis, 2023)

2.3.4.3. Swagger

Conforme a documentação oficial do *Swagger* (2024)*,* oSwagger é um conjunto de ferramentas de código aberto criadas em torno da Especificação OpenAPI que pode ajudar você a projetar, construir, documentar e consumir APIs REST.

O quadro xx indica as principais ferramentas do Swagger.

**Continuação do Quadro xx – Ferramentas Swager**

|  |  |
| --- | --- |
| Swagger Editor | Editor baseado em navegador onde você pode escrever definições OpenAPI. |
| Swagger UI | Renderiza definições OpenAPI como documentação interativa. |
| Swagger Codegen | Gera stubs de servidor e bibliotecas de cliente a partir de uma definição OpenAPI. |
| Swagger Editor Next (beta) | Editor baseado em navegador onde você pode escrever e revisar definições OpenAPI e AsyncAPI. |
| Swagger Core | Bibliotecas relacionadas a Java para criar, consumir e trabalhar com definições OpenAPI. |
| Swagger Parser | Biblioteca autônoma para analisar definições OpenAPI. |
| Swagger APIDom | Fornece uma estrutura única e unificadora para descrever APIs em várias linguagens de descrição e formatos de serialização. |

Fonte: Elaborado a partir da documentação oficial do Swagger (2024)

2.4. IOT – INTERNET DAS COISAS

IoT (Internet of Things) ou Internet das Coisas, foi um termo cunhado entre os anos 2008 e 2009 que representa a conectividade de dispositivos comuns do nosso dia a dia e industriais à rede de computadores para as diversas finalidades.

Durante esse período, o número de dispositivos conectados à Internet superou a população mundial. Com mais "coisas" conectadas à Internet do que pessoas no mundo, uma nova era estava chegando, e a Internet das Coisas nasceu. (Hanes et. al, 2017)

Neste capítulo serão abordados temas referentes às tecnologias IoT e aspectos adjacentes ao seu ecossistema.

**2.4.1 Conectividade**

Não há como falar de IoT sem falar em conectividade.

Podemos afirmar que conectividade em eletrônica ou computação é a capacidade de hosts, computadores, máquinas e todo equipamento eletrônico serem interligados de uma forma adequada para trocar dados e informações (em maior ou menor velocidade, eficiência etc.). (Ideali, 2021)

Na visão do autor o produto conectado é aquele que usa a internet para a sua conectividade, de modo a ser acessado e controlado à distância. (Sinclair, 2018)

Contudo para Carrion e Quaresma o objetivo da conectividade é maximizar a eficiência, conectar-se em diferentes pontos e em diversos dispositivos para fornecer uma experiência única ao usuário.

A conectividade pura e simples tem por objetivo maximizar a eficiência de um processo fixo em relação a problemas na qualidade da experiência de um usuário, principalmente em situações em que é necessário conectar-se em diferentes pontos em diversos dispositivos, com interpretação dos sinais de cada sensor para fornecer uma experiencia única ao usuário. (Carrion; Quaresma, 2019)

Sinclair afirma que nossas casas deixaram de ser “burras” e que todo este processo de conectividade está acontecendo com ou sem nossa intervenção.

Nossa casa deixou de ser burra para se tornar automatizada e, depois, inteligente, aglutinando segurança, gestão de energia e conveniência. A Internet das Coisas está acontecendo com ou sem você. E não porque é uma tecnologia legal... os clientes não se importam com a tecnologia dos fornecedores. A IoT está acontecendo pelo que ela proporciona. (Sinclair, 2018)

Desta forma, IoT deixou de ser algo exclusivo de um grupo de pessoas ou setores para se tornar parte da sociedade e de seu cotidiano. A aplicação desenvolvida neste projeto tem por finalidade efetuar a pesagem de alimentos e o controle sobre sua quantidade dentro do ambiente e contexto familiar que vai de encontro ao que o autor explana.

À medida que os produtos se tornam cada vez mais conectados, eles se comunicam e interagem com usuários e organizações em níveis novos e distintos. Dispositivos IoT estão alterando esferas diversas da sociedade, desde o gerenciamento doméstico e os cuidados com a saúde, à implementação nas indústrias, no planejamento urbano e de transportes, e na geração de energia, por exemplo. (Carrion; Quaresma, 2019).

**2.4.2. Ecossistema**

O IoT não é apenas uma tecnologia como também é composto de um ecossistema onde diversos dispositivos estão conectados e podem ou não conversar entre si. Dentro deste ecossistema há uma variedade grande de medidores, sensores, atuadores, sensores de rede e outros dispositivos.

Nos capítulos seguintes, exploraremos os principais componentes que formam este ecossistema.

2.4.2.1. Sensores

Sensores são amplamente utilizados no universo IoT e como diria Hanes et al (2017), mais especificamente, um sensor mede alguma quantidade física e converte essa leitura de medição em uma representação digital.

Sendo assim possível, com os dados extraídos destes dispositivos, através da interface de rede, criar aplicações de controle e gerência que utilizem estas informações para diversas finalidades.

Os sensores não se limitam a dados sensoriais semelhantes aos humanos. Eles podem medir qualquer coisa que valha a pena ser medida. De fato, eles são capazes de fornecer um espectro extremamente amplo de dados de medição ricos e diversificados com uma precisão muito maior do que os sentidos humanos; os sensores fornecem recursos sensoriais sobre-humanos. (Hanes et al, 2017)

No Quadro 9 é possível visualizar como são categorizados.

**Quadro 9** **– Categorias dos sensores**

|  |  |
| --- | --- |
| Ativo ou passivo | Os sensores podem ser categorizados com base no fato de produzirem uma saída de energia e normalmente exigirem uma fonte de alimentação externa (ativos) ou de simplesmente receberem energia e normalmente não exigirem fonte de alimentação externa (passivos). |
| Invasivo ou não invasivo | Os sensores podem ser categorizados com base no fato de um sensor ser parte do ambiente que está medindo (invasivo) ou externo a ele (não invasivo). |
| Com ou sem contato | Os sensores podem ser categorizados com base no fato de exigirem ou não contato físico com o que estão medindo (contato). |
| Absoluta ou relativa | Os sensores podem ser categorizados com base no fato de medirem em uma escala absoluta (absoluto) ou com base em uma diferença com um valor de referência fixo ou variável (relativo). |
| Área de aplicação | Os sensores podem ser categorizados com base no setor específico ou vertical em que estão sendo usados. |
| Como os sensores medem | Os sensores podem ser categorizados com base no mecanismo físico usado para medir a entrada sensorial (por exemplo, termoelétrico, eletroquímico, piezo resistivo, óptico, elétrico, mecânico de fluidos, fotoelástico). |
| O que os sensores medem | Os sensores podem ser categorizados com base em suas aplicações ou nas variáveis físicas que medem. |

Fonte: Elaborado a partir de (Hanes et al, 2017)

2.4.2.2. Atuadores

Os atuadores são complementos naturais dos sensores.

Atuadores podem receber um impulso ou comando de forma elétrica ou digital e assim atuar no mundo físico gerando movimento, criando alguma interação no ambiente que está inserido.

Os atuadores, por outro lado, recebem algum tipo de sinal de controle (geralmente um sinal elétrico ou comando digital) que aciona um efeito físico, geralmente algum tipo de movimento, força e assim por diante. (Hanes et al, 2017)

O Quadro 10 categoriza os atuadores.

**Quadro 10** **– Categorias dos atuadores**

|  |  |
| --- | --- |
| Tipo de movimento | Os atuadores podem ser classificados com base no tipo de movimento que produzem (por exemplo, linear, rotativo, um/dois/três eixos). |
| Potência | Os atuadores podem ser classificados com base em sua potência de saída (por exemplo, alta potência, baixa potência, micropotência) |
| Binário ou contínuo | Os atuadores podem ser classificados com base no número de saídas de estado estável. |
| Área de aplicação | Os atuadores podem ser classificados com base no setor específico ou vertical em que são usados. |
| Tipo de energia | Os atuadores podem ser classificados com base em seu tipo de energia. |
| Como os sensores medem | Os sensores podem ser categorizados com base no mecanismo físico usado para medir a entrada sensorial (por exemplo, termoelétrico, eletroquímico, piezo resistivo, óptico, elétrico, mecânico de fluidos, fotoelástico). |
| O que os sensores medem | Os sensores podem ser categorizados com base em suas aplicações ou nas variáveis físicas que medem. |

Fonte: Elaborado a partir de (Hanes et al, 2017, tradução nossa)

2.4.2.3. Comunicações

A escolha da forma de comunicação de um dispositivo IoT dentro de uma infraestrutura é muito importante. Nem sempre é necessário projetar uma estrutura superdimensionada em razão de que dispositivos IoT consomem pouco tráfego de rede e isso quer dizer que por exemplo, um dispositivo que apenas faz a leitura de temperatura de um ambiente ao longo do tempo, não necessita de uma comunicação wireless 802.11ac (5 GHz), e que seria muito melhor a comunicação wireless 802.11g (2,4 GHz) devido a capacidade da frequência de onda empregada nessa tecnologia ter maior capacidade de atravessar obstáculos.

Para a conectividade sem fio, a forma de onda portadora é lançada como uma onda eletromagnética no "espaço livre" do dispositivo transmissor para o dispositivo receptor. Na realidade, o "espaço livre" inclui obstáculos físicos, como ar, solo, edifícios, paredes, pessoas, árvores, umidade e chuva etc., que afetam a fidelidade da onda que chega à antena receptora. (Ramgir, 2019)

Como podemos ver na Figura 1 o comprimento de onda na frequência de 2,4 GHz é maior em relação à frequência de 5 GHz e isso quer dizer que a uma rede wireless 802.11g tem um alcance maior, portanto, melhor e mais atrativo para aplicações IoT.

**Texto

Descrição gerada automaticamenteFigura 1 – Forma de onda**

2.4.2.4. Protocolo MQTT

O Message Queue Telemetry Transport (MQTT) é um protocolo de comunicação de muitos para muitos com um mediador de mensagens chamado de broker. Existem editores que enviam mensagens para os tópicos no broker, e há subscritores que recebem mensagens dos tópicos que subscreveram. Um nó pode ser um editor e um assinante ao mesmo tempo. A figura 2 demonstra de forma simplificada o funcionamento da estrutura. (Oner, 2021)

A figura xx ilustra de forma simples a arquitetura do protocolo MQTT.

Diagrama

Descrição gerada automaticamente**Figura 2 – Protocolo MQTT**

Fonte: Acervo do autor

Como Oner cita, para publicar uma mensagem são necessários vários parâmetros, incluindo o cliente MQTT, o nome do tópico e os dados a serem publicados. Um parâmetro especifica a qualidade do serviço, ou QoS. Há três opções válidas para QoS:

1. QoS nível 0: Significa que não há garantia de entrega aos assinantes. Os assinantes(subscribers) não têm de acusar a recepção da mensagem.
2. QoS nível 1: Garante a entrega, mas os assinantes podem receber várias cópias da mensagem.
3. QoS nível 2: Os assinantes recebem exatamente uma cópia da mensagem.

E continua com a informação de que, selecionamos o QoS, de acordo com os requisitos do aplicativo. Por exemplo, se pular uma leitura não for tão importante, podemos selecionar QoS-0. A QoS-1 e o QoS-2 são obtidas por meio de mensagens adicionais de confirmação (ACK), que introduzem alguma sobrecarga na comunicação.

De acordo com *Steve’s Internet Guide* (2024), o MQTT é um protocolo leve de publicação/assinatura de mensagens projetado para telemetria M2M (máquina a máquina) em ambientes de baixa largura de banda. Ele foi projetado por Andy Stanford-Clark (IBM) e Arlen Nipper em 1999 para conectar sistemas de telemetria de oleodutos por satélite e embora tenha começado como um protocolo proprietário, foi liberado sem royalties em 2010 e tornou-se um padrão OASIS em 2014.

E conclui que, o MQTT, está se tornando rapidamente um dos principais protocolos para implementações de IOT (Internet das Coisas).

2.4.2.5. Broker – Mosquitto

De acordo com a documentação oficial do *Mosquito* (2024), o Eclipse Mosquitto é um broker de mensagens de código aberto (licenciado pela EPL/EDL) que implementa as versões 5.0, 3.1.1 e 3.1 do protocolo MQTT. O Mosquitto é leve e adequado para uso em todos os dispositivos, desde computadores de placa única de baixa potência até servidores completos.

O protocolo MQTT fornece um método leve de realizar mensagens usando um modelo de publicação/subscrição. Isso o torna adequado para mensagens da Internet das Coisas, como sensores de baixa potência ou dispositivos móveis, como telefones, computadores incorporados ou microcontroladores. O projeto Mosquitto também fornece uma biblioteca C para implementar clientes MQTT e os populares clientes MQTT de linha de comando mosquitto\_pub e mosquitto\_sub. O Mosquitto faz parte da Eclipse Foundation e é um projeto iot.eclipse.org.

2.4.2.6. Porque Mosquitto

De acordo com a documentação oficial da *Eclipse Foundation* (2024), normalmente, a implementação atual do Mosquitto tem um executável da ordem de 120 KB que consome cerca de 3 MB de RAM com 1.000 clientes conectados. Há relatos de testes bem-sucedidos com 100.000 clientes conectados a taxas de mensagens modestas.

Além de aceitar conexões de aplicativos clientes MQTT, o Mosquitto tem uma ponte que permite que ele se conecte a outros servidores MQTT, incluindo outras instâncias do Mosquitto. Isso permite a construção de redes de servidores MQTT, passando mensagens MQTT de qualquer local da rede para qualquer outro, dependendo da configuração das pontes.

2.5. Arduino – Uma curta história

Em 2005, com base no trabalho de Hernando Barragán (criador do Wiring), Massimo Banzi e David Cuartielles criaram o Arduino, um dispositivo programável fácil de utilizar para projetos de design artístico interativo, no Interaction Design Institute Ivrea em Ivrea, Itália.

Eles queriam um dispositivo que fosse simples, fácil de ligar a várias coisas (como relés, motores e sensores) e fácil de programar. Também precisava de ser barato, uma vez que os estudantes e os artistas não são conhecidos por terem muito dinheiro disponível. (Hughes, 2016)

Desde então, o Arduino cresceu em várias direções diferentes, com algumas versões ficando menores que a original e outras maiores. Cada uma delas tem um nicho específico a ser preenchido. O elemento comum entre todas elas são a biblioteca AVR-GCC de tempo de execução do Arduino, que é fornecida com o ambiente de desenvolvimento do Arduino, e o firmware do carregador de inicialização integrado que vem pré-carregado no microcontrolador de cada placa Arduino.

Embora uma placa Arduino seja, como afirma a equipe Arduino, apenas uma placa de desenvolvimento Atmel AVR básica, é o ambiente de software Arduino que a diferencia. Essa é a experiência comum para todos os usuários do Arduino e a pedra angular do conceito do Arduino. (Hughes, 2016)

2.5.1. A maneira arduíno de ser

A filosofia do Arduino se baseia em fazer projetos em vez de falar sobre eles. É uma busca constante por maneiras mais rápidas e poderosas de criar protótipos melhores. Exploramos muitas técnicas de prototipagem e desenvolvemos formas de pensar com nossas mãos. (Banzi, 2009)

A engenharia clássica se baseia em um processo rigoroso para ir de A a B; a maneira arduíno de ser se deleita com a possibilidade de se perder no caminho e, em vez disso, encontrar C.

Esse é o processo de conserto de que tanto gostamos: brincar com o meio de uma forma aberta e encontrar o inesperado. Nessa busca por maneiras de criar protótipos melhores, também selecionamos vários pacotes de software que permitem o processo de manipulação constante da mídia de software e hardware. (Banzi, 2009)

2.5.2. ESP32

Desde o lançamento do primeiro chip ESP32, a Espressif Systems ampliou a família com novos designs para diferentes finalidades. Atualmente, a empresa tem mais de 200 SoCs e módulos diferentes em seu inventário. Embora seja impossível discutir cada um deles um a um, podemos falar sobre a família de produtos ESP32 em geral para entender os casos de uso pretendidos. É sempre uma boa ideia verificar o que precisamos e o que está disponível no arsenal antes de iniciar um novo projeto de IoT. (Oner, 2023)

Essa série é o primeiro produto da família de produtos ESP32 e dá nome a toda a família de SoCs (System on a Chip – Sistema no Chip). Ela contém as variantes mais comuns de chips da família de produtos ESP32. A figura 2 demonstra o diagrama de blocos funcionais em sua folha de dados.

Diagrama

Descrição gerada automaticamente**Figura 2 – Diagrama de blocos ESP32**

Fonte: Acervo do autor

3. Metodologia DA PESQUISA

A metodologia adotada no desenvolvimento do trabalho intitulado **"SISTEMA GERENCIADOR DE GRÃOS PARA DESPENSA DE ALIMENTOS FAMILIAR"** apresenta uma natureza exploratória e descritiva, com base em abordagens qualitativas. Essa escolha metodológica se justifica pela necessidade de compreender, detalhar e expandir os conceitos relacionados ao uso da IoT (Internet das Coisas) no gerenciamento doméstico de alimentos, abordando tanto os aspectos técnicos quanto os contextos de aplicabilidade social e sustentabilidade. Na figura 4 é demonstrado o fluxograma de processos deste trabalho.

Diagrama

Descrição gerada automaticamente**Figura 4 – Fluxograma de processos**

Fonte: Acervo do autor.

A pesquisa descritiva foi utilizada para mapear as tecnologias envolvidas na construção do sistema, como sensores de pesagem analógicos, conversor digital, conectividade sem fio utilizando protocolos MQTT, algoritmos para controle de estoque e otimização do consumo. Através dessa abordagem, foi possível apresentar detalhadamente as funcionalidades do sistema, desde a captação de dados pelos dispositivos IoT até a integração com APIs e o armazenamento de informações no banco de dados. Já a pesquisa exploratória foi essencial para identificar novas possibilidades e caminhos inovadores, expandindo o potencial de aplicação do sistema para contextos maiores, como comunidades ou instituições que enfrentam desafios na gestão eficiente de alimentos.

O método qualitativo adotado nesta pesquisa permitiu transformar dados em conceitos e ideias que sustentam o desenvolvimento do sistema. A coleta de dados baseou-se em **fontes primárias** e **fontes secundárias**, abrangendo uma ampla revisão bibliográfica composta por artigos científicos, livros técnicos e manuais especializados. Entre os temas abordados na literatura, destacam-se a automação com IoT, a gestão eficiente de estoques, e o combate ao desperdício alimentar. Essas leituras não apenas embasaram o trabalho, mas também guiaram as decisões técnicas tomadas ao longo do desenvolvimento do projeto, como a escolha de sensores de pesagem e a utilização de um broker MQTT para comunicação em tempo real.

A escolha do tema foi motivada pela relevância social e ambiental da redução do desperdício de alimentos, que representa um dos grandes desafios globais atuais. Além disso, o desenvolvimento de um sistema automatizado voltado ao uso doméstico alia inovação tecnológica com impacto direto na sustentabilidade, incentivando hábitos de consumo mais conscientes. A pesquisa bibliográfica foi conduzida de forma sistemática, priorizando autores renomados, estudos recentes e manuais técnicos, em 99% dos casos em língua estrangeira (Inglês), que abordassem a implementação prática de tecnologias emergentes tanto no campo do IoT e da automação como na área da ciência da computação.

Na prática, a pesquisa foi aplicada de maneira a unir teoria e experimentação. O trabalho incluiu o desenvolvimento de protótipos que integraram sensores de pesagem analógicos com conversor digital a uma plataforma IoT, ESP32, utilizando o protocolo MQTT para comunicação com um broker, subscrevendo dados da pesagem dos alimentos. Esse sistema foi implementado com foco na coleta e processamento de dados em tempo real, permitindo o registro do peso dos grãos armazenados e a geração de alertas sobre reabastecimento. A interface do sistema foi projetada em React Native para garantir acessibilidade e usabilidade, permitindo que os usuários acompanhem os estoques de maneira intuitiva tanto nos sistemas Android como iOS.

A validação do sistema foi realizada por meio de experimentos práticos que simularam o uso real em ambiente doméstico. Durante os testes, foram avaliados a precisão dos sensores, a confiabilidade da transmissão de dados via MQTT e a eficácia da API em processar e armazenar informações no banco de dados PostgreSQL. Adicionalmente, a usabilidade da interface foi analisada com base em critérios de clareza e eficiência na apresentação dos dados aos usuários.

Essa abordagem metodológica não apenas resultou em uma solução prática e funcional para o gerenciamento de alimentos em escala doméstica, mas também evidenciou o potencial de tecnologias similares em contribuir para iniciativas globais para evitar desperdício de alimentos e segurança alimentar. A união entre pesquisa descritiva, exploração tecnológica e métodos qualitativos possibilitou o desenvolvimento de um sistema que vai além do benefício imediato, promovendo sustentabilidade e consumo consciente em um contexto mais amplo. Por fim, o projeto demonstra como a combinação de inovação tecnológica e pesquisa bem fundamentada pode gerar impacto positivo na vida das pessoas e no meio ambiente.

4. projetando

Neste capítulo, serão abordados os temas referentes ao desenvolvimento da aplicação, fluxos, testes e análises.

4.1. FLUXOGRAMA

Neste capítulo serão apresentados os fluxogramas que ilustram o funcionamento detalhado do sistema. Os fluxogramas foram desenvolvidos para descrever os processos envolvidos na integração entre os dispositivos IoT, comunicação via broker MQTT, APIs para processamento de dados e a interação dos usuários com o sistema. Cada fluxograma tem como objetivo facilitar a compreensão das etapas e das interações entre os diferentes componentes do sistema, garantindo clareza no fluxo de informações e operações.

Os fluxogramas contemplam os seguintes aspectos:

* **Fluxograma do Sistema Geral:** Demonstra a integração e o fluxo completo desde a coleta de dados pelo dispositivo IoT (ESP32) até o gerenciamento de estoques e produtos na aplicação mobile.
* **Fluxograma do Banco de Dados:** Detalha a estrutura de tabelas e os relacionamentos entre entidades no banco de dados PostgreSQL, assegurando a integridade dos dados.
* **Fluxograma de Processos Específicos:** Descreve etapas críticas, como a captação de dados pela balança, o envio ao broker MQTT, o processamento pelas APIs, e o armazenamento no banco de dados.

Esses fluxogramas não apenas destacam a lógica por trás do sistema, mas também oferecem uma visão clara das interações e dependências entre os componentes, garantindo suporte tanto para o desenvolvimento quanto para a manutenção futura do sistema.

4.1.1. Fluxograma do sistema

O sistema foi concebido para oferecer uma integração fluida entre dispositivos IoT, comunicação em tempo real e um banco de dados centralizado, permitindo um gerenciamento eficiente de estoques e produtos. Com uma arquitetura robusta e escalável, o sistema utiliza o broker MQTT Mosquitto como mediador na troca de informações, enquanto API’s garantem a integração e o armazenamento dos dados em um banco de dados PostgreSQL. A seguir, o detalhamento do fluxo operacional do sistema.

A figura xx ilustra o fluxograma geral do sistema.

**Figura xx – Fluxograma do geral**

Diagrama

Descrição gerada automaticamente

Fonte: Acervo do autor

4.1.2. Captação de Dados pelo Dispositivo IoT (ESP32 e Balança Integrada)

Neste capítulo serão abordados o processo da captação de dados pelo disposto IoT.

* **Descrição do Processo:**
  + O **ESP32**, um microcontrolador configurado para leitura contínua dos dados de peso da balança, monitora em tempo real qualquer alteração no peso dos produtos.
  + Sempre que há uma mudança significativa acima da margem de tolerância de 2% no peso detectado pela balança, o ESP32 processa e formata os dados em um protocolo MQTT.
* **Dados Transmitidos:**
  + **Peso Atualizado:** Valor numérico representando o peso medido em tempo real.
  + **Identificador do Produto:** Um UUID que identifica unicamente o produto.
  + **Timestamp:** A data e hora da medição, assegurando rastreabilidade.
* **Benefícios:**
  + Minimiza o tráfego desnecessário, publicando apenas mudanças relevantes.
  + Garante baixa latência na transmissão dos dados para o próximo componente do fluxo.
* **QoS 1:**
  + No QoS 1 do MQTT, o foco está em garantir a entrega da mensagem pelo menos uma vez ao receptor. Quando uma mensagem é publicada com QoS 1, o remetente mantém uma cópia da mensagem até receber um pacote PUBACK do receptor, confirmando o recebimento bem-sucedido. Se o remetente não receber o pacote PUBACK em um prazo razoável, ele retransmite a mensagem para garantir sua entrega.

A figura xx ilustra o funcionamento da comunicação da balança com o broker através do QoS 1.

**Figura xx – QoS 1**

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente

Fonte: Acervo do autor

4.1.3. Comunicação com o Broker MQTT

Neste capítulo serão abordados o funcionamento do broker MQTT.

* **Papel do Broker MQTT:**
  + O broker Mosquitto atua como o núcleo da comunicação em tempo real, oferecendo um mecanismo eficiente de publicação e assinatura de mensagens entre o ESP32 e as APIs do sistema.
* **Fluxo de Comunicação:**
  + O **ESP32** publica os dados capturados em tópicos específicos no broker MQTT, permitindo que consumidores (APIs) assinem esses tópicos para receber atualizações instantâneas.
  + Mensagens seguem o modelo **publish/subscribe**, reduzindo o acoplamento entre os componentes.
* **Escalabilidade e Resiliência:**
  + O broker Mosquitto suporta múltiplos dispositivos IoT e consumidores simultaneamente, sendo capaz de gerenciar picos de mensagens.
  + Implementação com suporte a autenticação e criptografia garante segurança dos dados.
  + O broker Mosquitto tem um executável de 120 KB que consome cerca de 3 MB de RAM com 1.000 clientes ativos.

4.1.4. Consulta e processamento de dados pela API da regra de negócios

O papel da API de regra de negócios e comunicação com o broker Mosquitto é servir como uma camada intermediária, projetada com base em uma arquitetura orientada a micro serviços, para interagir eficientemente com o broker MQTT. Essa API é responsável por consultar os tópicos publicados no broker em intervalos regulares ou sob demanda, garantindo um fluxo contínuo de dados entre os dispositivos IoT e o sistema principal.

Também é responsável por efetuar a comunicação com a aplicação mobile onde são efetuados os gerenciamentos de estoque, produtos, categorias, bem como a autenticação dos usuários.

No fluxo operacional, a API é implementada como um micro serviço independente, o que permite sua escalabilidade e isolamento funcional. Ela assina os tópicos relevantes no broker MQTT e consome as mensagens enviadas por dispositivos, como o ESP32. Durante esse processo, são realizadas validações iniciais nos dados recebidos, incluindo verificações de formato, consistência e integridade, utilizando-se de tratamento de exceções. Após o tratamento, os dados validados são transmitidos para outro micro serviço, diretamente para a API da base de dados, que é responsável por persistir as informações.

Essa abordagem baseada em micro serviços apresenta diversas vantagens técnicas. Uma das quais posso citar é o desacoplamento entre os dispositivos IoT e a API “principal” que simplifica a manutenção e evolução do sistema, uma vez que cada componente pode ser atualizado ou escalado independentemente. Além disso, a API reduz significativamente a carga sobre o banco de dados, processando apenas as alterações relevantes antes de transmiti-las. Esse design também facilita a integração com outros componentes ou serviços, como filas de mensagens (RabbitMQ, Kafka) ou sistemas de monitoramento em tempo real.

Por fim, o uso de micro serviços permite a adoção de práticas modernas de DevOps, como pipelines de CI/CD para atualizações frequentes, monitoramento detalhado de métricas e logs, e resiliência aprimorada, com capacidade de lidar com falhas parciais sem comprometer o sistema como um todo. Essa arquitetura proporciona não apenas maior eficiência operacional, mas também escalabilidade e flexibilidade para atender às demandas crescentes do sistema.

Abaixo podemos visualizar a estrutura resumida em tópicos.

* **Papel da API**:
  + Atua como uma camada intermediária, baseada em uma arquitetura de micro serviços, para interagir com o broker MQTT.
  + Consulta tópicos do broker MQTT de forma regular ou sob demanda, garantindo um fluxo contínuo de dados.
  + Atende às requisições da aplicação mobile para consultar dados, inserir e atualizar informações.
* **Fluxo Operacional**:
  + Implementada como um micro serviço independente, a API é responsável por assinar tópicos relevantes e consumir mensagens enviadas por dispositivos IoT, como o ESP32.
  + Realiza validações iniciais nos dados recebidos, incluindo:
  + Verificação de formato e consistência.
  + Garantia de integridade para evitar propagação de erros.
  + Transmite os dados processados para a API principal ou para outros micros serviços, que gerenciam o armazenamento no banco de dados.
* **Vantagens Técnicas**:
  + **Desacoplamento**: A arquitetura baseada em micro serviços separa os dispositivos IoT da API principal, permitindo atualizações e manutenção independentes.
  + **Redução de Carga no Banco de Dados**: Processa apenas alterações relevantes antes de transmitir os dados, otimizando a eficiência do sistema.
  + **Escalabilidade**: Cada componente do sistema pode ser escalado individualmente para atender a demandas crescentes.
* **Integração com Outros Sistemas**:
  + Suporte a filas de mensagens como RabbitMQ ou Kafka para maior confiabilidade e paralelismo.
  + Compatibilidade com sistemas de monitoramento em tempo real.
* **DevOps e Resiliência**:
  + Adaptação a pipelines CI/CD para atualizações contínuas.
  + Monitoramento detalhado de logs e métricas para diagnóstico e otimização.
  + Capacidade de lidar com falhas parciais sem comprometer o funcionamento do sistema completo.

4.1.6. Interação e Administração pela Aplicação Mobile

A **interação e administração pela aplicação mobile**, desenvolvida em **React Native**, foi projetada como parte de uma arquitetura orientada a **micro serviços**, garantindo modularidade, escalabilidade e flexibilidade no sistema. Essa aplicação atua como a interface principal para administradores e usuários, permitindo o gerenciamento e consulta de estoques e produtos de forma eficiente e segura, com foco em atualizações em tempo real.

Para administradores, a aplicação oferece funcionalidades avançadas, como a adição de novos usuários, estoques e produtos, além do gerenciamento granular de permissões de acesso. Também possibilita a visualização de históricos detalhados de alterações no peso dos produtos, fornecendo rastreabilidade completa. Para usuários finais, a aplicação permite a consulta de estoques vinculados ao seu perfil e o acompanhamento em tempo real das mudanças nos produtos, garantindo transparência e agilidade na gestão.

A comunicação entre a aplicação e o sistema é baseada em um fluxo integrado com **micro serviços**, onde a aplicação mobile interage diretamente com a API principal e outros serviços especializados. Cada funcionalidade crítica é tratada por micro serviços independentes, responsáveis por operações específicas, como o gerenciamento de usuários, produtos ou estoques. Essa abordagem assegura que os dados exibidos no aplicativo sejam sempre consistentes e reflitam os registros mais recentes do banco de dados. Além disso, a separação de responsabilidades entre micro serviços permite maior resiliência do sistema, uma vez que falhas em um serviço não comprometem o funcionamento global.

A aplicação mobile aproveita as vantagens técnicas dessa arquitetura, como maior flexibilidade para implementar novos recursos e escalabilidade para atender a um número crescente de usuários. A interface amigável e responsiva, projetada em **React Native**, é acessível em dispositivos móveis, garantindo alta performance e experiência uniforme. As atualizações em tempo real, facilitadas por comunicações assíncronas e eventos do sistema baseados em filas, como RabbitMQ ou Kafka, permitem aos usuários tomar decisões rápidas e informadas, alinhadas às necessidades de gestão e administração modernas.

4.2. fluxograma da aplicação mobile

4.3. fluxograma do banco de dados

O banco de dados foi projetado para atender as necessidades de gerenciamento de usuários, estoques, produtos e suas respectivas categorias/subcategorias.

A figura xx ilustra as entidades da base de dados.

**Figura xx – Base de dados PostgresSql**

Diagrama, Desenho técnico, Esquemático

Descrição gerada automaticamente

Fonte: Acervo do autor

4.3.1. Funcionamento do banco de dados

Abaixo estão os detalhes do funcionamento e fluxo do banco de dados:

4.3.1.1. Criação de Usuário (Administrador)

A aplicação inicia com o registro de um usuário administrador (Admin), responsável por gerenciar o sistema e conceder acesso a outros usuários. O processo segue as seguintes etapas:

* **Registro de Dados do Administrador:**
  + Os dados essenciais do administrador, como Nome, Email, Senha, e Sobrenome, são cadastrados na tabela User.
* **Definição de Permissão:**
  + O administrador recebe automaticamente a permissão correspondente ao papel de Admin, que permite acesso total às funcionalidades do sistema, incluindo a criação de usuários.
* **Status Ativo:**
  + O campo Ativo da tabela User é configurado com o valor padrão de 1, indicando que o administrador está habilitado para operar no sistema.
  + Essa configuração inicial garante que o primeiro usuário criado no sistema tenha os privilégios necessários para gerenciar todos os recursos e entidades do banco de dados.

4.3.1.2. Gestão de Usuários

A gestão de usuários é uma funcionalidade central no sistema, permitindo ao administrador gerenciar o acesso e as permissões dos usuários. O processo ocorre da seguinte forma:

* **Adição de Novos Usuários:**
  + O administrador tem a capacidade de registrar novos usuários no sistema.
  + Durante o registro, é possível definir a permissão do usuário.
* **Admin:**
  + Usuários com acesso total, permitindo que também gerenciem outros usuários e estoques.
* **User:**
  + Usuários com acesso limitado, restrito à gestão de produtos e consulta de estoques associados.
* **Vinculação de Usuários à Tabela User:**
  + As informações básicas do usuário, como Nome, Email, Sobrenome, Senha, e Status Ativo, são armazenadas na tabela User.
  + O campo Ativo, por padrão, será definido como 1, indicando que o usuário está habilitado no sistema.
* **Gestão de Estoques Associados:**
  + Cada usuário é vinculado a um ou mais estoques por meio da tabela intermediária **UserEstoque**.

A tabela **UserEstoque** faz o vínculo da tabela **Estoque** com a tabela **User** e possui os atributos:

* **UserEstoqueId**: Identificação da tabela UserEstoqueId.
* **UserId**: Identifica o usuário que tem acesso ao estoque.
* **EstoqueId**: Identifica o estoque associado.
* **EstoqueId**: Identifica o estoque associado.
* **Permissão**: Define o nível de acesso do usuário ao estoque (Admin ou User).
* **Owner**: Indica se o usuário é proprietário do estoque.
* **Ativo**: Controla se a associação entre usuário e estoque está ativa.

Segurança e Controle:

* O sistema assegura que apenas administradores possam adicionar novos usuários ou alterar permissões.
* Todas as operações relacionadas a usuários e estoques são registradas e rastreáveis para garantir a consistência e a integridade dos dados.
* Essa abordagem de gestão garante que o sistema seja flexível para atender diferentes cenários organizacionais, com controle granular sobre os acessos e as permissões dos usuários.
* O sistema assegura que apenas administradores possam adicionar novos usuários ou alterar permissões.
* Todas as operações relacionadas a usuários e estoques são registradas e rastreáveis para garantir a consistência e a integridade dos dados.
* Essa abordagem de gestão garante que o sistema seja flexível para atender diferentes cenários organizacionais, com controle granular sobre os acessos e as permissões dos usuários.

4.3.1.3. Gestão de Estoques

A gestão de estoques é um componente fundamental do sistema, permitindo que tanto administradores quanto usuários criem e gerenciem estoques. O processo é descrito detalhadamente a seguir.

Tanto administradores quanto usuários com permissões adequadas podem criar estoques.

Cada estoque é identificado como uma entidade única, com os seguintes atributos armazenados na tabela **Estoque**:

* **EstoqueId**: Identifica o id estoque.
* **Nome**: Nome do estoque, que deve ser descritivo e facilitar a identificação.
* **Descrição**: Detalhes sobre o estoque, permitindo registrar informações relevantes.
* **DataCriacao**: Data e hora em que o estoque foi criado.
* **Ativo**: Indica se o estoque está ativo no sistema (padrão: 1).

4.3.1.3. Relacionamento entre Estoques e Usuários

A tabela **UserEstoque** faz o vínculo da tabela **Estoque** com a tabela **User** e possui os atributos:

* **UserEstoqueId**: Identificação da tabela UserEstoqueId.
* **UserId**: Identifica o usuário que tem acesso ao estoque.
* **EstoqueId**: Identifica o estoque associado.
* **EstoqueId**: Identifica o estoque associado.
* **Permissão**: Define o nível de acesso do usuário ao estoque (Admin ou User).
* **Owner**: Indica se o usuário é proprietário do estoque.
* **Ativo**: Controla se a associação entre usuário e estoque está ativa.

4.3.1.3.1. Regras de acesso e controle

O sistema assegura que apenas usuários com permissões específicas possam criar ou alterar estoques.

Quando um estoque é excluído pelo proprietário (indicado pelo campo Owner na tabela UserEstoque), todas as associações vinculadas e produtos relacionados ao estoque são removidos automaticamente. Em termos práticos, o estoque não é excluído de fato e sim desabilitado para que se possa manter o registro das informações.

Cada estoque pode ser compartilhado entre múltiplos usuários, permitindo colaboração entre administradores e usuários no gerenciamento dos produtos.

Essa estrutura e gestão de estoques oferece flexibilidade para diferentes cenários, desde pequenos inventários pessoais até estoques colaborativos de larga escala, garantindo controle granular e segurança na administração.

4.3.1.4. Produtos no Estoque

Os produtos são os elementos principais no gerenciamento de estoques, sendo cada produto vinculado diretamente a um estoque específico. O funcionamento é descrito a seguir:

* **Relacionamento entre Estoque e Produtos:**
  + Um **estoque** pode conter diversos produtos, permitindo a gestão de múltiplos itens dentro de um mesmo inventário.
  + Cada **produto** pertence exclusivamente a um único estoque, garantindo que as informações sejam organizadas e rastreáveis.
* **Armazenamento de Produtos:**
  + Os dados dos produtos são registrados na tabela **Produto**, que inclui:
    - **Nome:** Nome do produto, facilitando sua identificação.
    - **QuantidadeIn:** Quantidade de entrada do produto no estoque, armazenada com precisão numérica.
    - **QuantidadeOut:** Quantidade de saída do produto, útil para rastrear o consumo ou vendas.
    - **DataIn:** Data e hora da última entrada do produto no estoque.
    - **DataOut:** Data e hora da última saída do produto.
    - **CatSubId:** Identifica a categoria ou subcategoria vinculada ao produto (opcional).
    - **EstoqueId:** Identifica o estoque ao qual o produto pertence.
    - **Ativo:** Indica se o produto está ativo no sistema (padrão: 1).
* **Rastreabilidade:**
* O sistema registra automaticamente as alterações de peso ou quantidade dos produtos, garantindo que o histórico de movimentações seja rastreável e acessível.

4.3.1.5. Categorias e Subcategorias de Produtos

Para melhor organização, os produtos podem ser classificados em categorias e subcategorias. A estrutura de categorização segue as seguintes diretrizes:

* Classificação de Produtos:
  + Os produtos podem ser categorizados usando a tabela intermediária CatSub, que estabelece o vínculo entre as tabelas Categoria e SubCategoria.
  + Um produto pode ter:
    - Uma categoria opcional: Definida pela chave estrangeira CategoriaId.
    - Uma subcategoria opcional: Associada à categoria e definida pela chave estrangeira SubCategoriaId.
* Flexibilidade de Categorização:
  + A categorização é opcional, permitindo que produtos sejam registrados sem qualquer categoria ou subcategoria.
  + Quando uma categoria é definida, a subcategoria vinculada também pode ser especificada, mas não é obrigatória.
* Armazenamento de Categorias e Subcategorias:
  + Categoria: Contém informações como nome, descrição, e status de ativo.
  + SubCategoria: Similar à categoria, mas associada a uma categoria específica, permitindo maior granularidade na classificação.

4.3.1.6. Estrutura de Relacionamento

O sistema foi projetado para garantir flexibilidade e escalabilidade, com relacionamentos claros e bem definidos entre as tabelas. Os principais relacionamentos são descritos a seguir:

* **Usuários e Estoques (1:N):**
  + Cada usuário pode gerenciar múltiplos estoques por meio da tabela **UserEstoque**.
  + Essa relação permite que administradores e usuários tenham permissões distintas sobre os estoques associados.
* **Estoques e Produtos (1:N):**
  + Um estoque pode conter múltiplos produtos, mas cada produto está vinculado a apenas um estoque.
  + Esse relacionamento garante que os produtos sejam organizados de forma clara dentro de cada estoque.
* **Produtos, Categorias e Subcategorias (0..1:N):**
  + Cada produto pode opcionalmente ser associado a uma categoria.
  + Se categorizado, o produto também pode ser vinculado a uma subcategoria, que deve estar associada à categoria principal.
  + A flexibilidade do relacionamento permite que produtos sejam registrados sem categorias ou subcategorias, simplificando a inserção de dados quando necessário.

5. LEVANTAMENTO DE REQUISITOS

Dentro da engenharia de software temos como um dos principais pilares da construção de um sistema o levantamento de requisitos. Este capítulo trará informações sobre os requisitos funcionais e não funcionais das API’s do sistema.

4.1. API DE CONSULTA AO BROKER E REGRA DE NEGÓCIOS

Neste capítulo serão abordados os requisitos funcionais da API de consulta ao broker e de regra de negócios. Esta API é responsável por fazer a subscrição no tópico de um produto do estoque e ficar monitorando se há novos dados de pesagem dos alimentos.

Ao receber novas informações esta API faz o tratamento dos dados e os envia para a API responsável pela manipulação da base dados.

5.1.1 Requisitos funcionais

No quadro xx pode-se visualizar os requisitos funcionais da API de consulta ao broker e de regra de negócios.

**Quadro 12 – Requisitos funcionais**

|  |  |  |  |
| --- | --- | --- | --- |
| **Nº** | **Nome** | **Classificação** | **Descrição** |
| **RF 01** | Login do sistema | Segurança | O sistema deverá dispor meios de autenticação para usuários e administradores. |
| **RF 02** | Cadastro, alteração e exclusão de usuários | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de usuários. |
| **RF 03** | Cadastro, alteração e exclusão de produtos | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de produtos. |
| **RF 04** | Cadastro, alteração e exclusão de categorias | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de categorias. |
| **RF 05** | Cadastro, alteração e exclusão de subcategorias | Usabilidade | O sistema deve possibilitar o cadastro de subcategorias de um produto. |
| **RF 06** | Cadastro, alteração e exclusão de estoque | Usabilidade | O sistema deve permitir o cadastro, alteração e exclusão de itens no estoque, atualizando automaticamente as quantidades disponíveis. |
| **RF 07** | Controle de permissões | Segurança | O sistema deverá dispor de um controle de permissões para definir os níveis de acesso de usuários (ex.: administrador, operador). |
| **RF 08** | Exibição de itens | Usabilidade | O sistema deverá dispor de uma rotina que exiba todos os itens cadastrados. |
| **RF 09** | Quantidade | Usabilidade | O sistema deverá dispor de uma rotina que exiba a quantidade em Kg de cada item cadastrado. |
| **RF 10** | Setar quantidade mínima | Usabilidade | O sistema deverá dispor de uma rotina que permita setar a quantidade mínima de um produto. |
| **RF 11** | Alertas | Confiabilidade | O sistema deverá dispor de uma rotina que emita alertas de acordo com a quantidade mínima configurada. |
| **RF 12** | Entrada e saída de itens | Rastreabilidade | O sistema deverá dispor de uma rotina que registre a data de quando um item é adicionado e quando é retirado. |
| **RF 13** | Número de refeições | Usabilidade | O sistema deverá dispor de uma rotina que calcule o número de refeições restantes que cada item possui. |
| **RF 14** | Segurança de senhas | Segurança | O sistema deverá exigir senhas com no mínimo 8 caracteres, incluindo letras maiúsculas e números. |
| **RF 15** | Armazenamento de senhas | Segurança | O sistema deverá gravar as senhas de forma criptografada no banco de dados. |

Fonte: Acervo do autor.

5.1.2 Requisitos não funcionais

No quadro xx pode-se visualizar os requisitos funcionais da API de consulta ao broker e de regra de negócios.

**Quadro 11 – Requisitos não funcionais**

|  |  |  |  |
| --- | --- | --- | --- |
| **Nº** | **Nome** | **Classificação** | **Descrição** |
| **RNF 01** | Multiplataforma | Portabilidade | O sistema deve ser multiplataforma, permitindo ao usuário maior flexibilidade de acesso e produtividade. |
| **RNF 02** | Consistência de dados | Confiabilidade | O sistema deve ser desenvolvido para que as informações inseridas e consultadas sejam consistentes. |
| **RNF 03** | Segurança do sistema | Segurança | O sistema deve adotar política e medidas de segurança. |
| **RNF 04** | Interface amigável | Usabilidade | O sistema deve ser desenvolvido de forma que o usuário tenha facilidade em sua utilização. |
| **RNF 05** | Consulta e gravação de dados | Performance | O sistema deve possuir boa performance para consulta e gravação no banco de dados. |
| **RNF 06** | Manutenção | Manutenibilidade | O sistema deve ser de fácil manutenibilidade, usando padrão de projetos para permitir fácil implementação de novas funcionalidades e correção de erros. |
| **RNF 07** | Servidor | Confiabilidade | O sistema deve ser hospedado em servidor cloud e ter backups diários da base de dados. |
| **RNF 08** | Programação backend | Padrões | A programação backend do sistema deve ser construída em C#. |
| **RNF 09** | Programação frontend | Padrões | A programação frontend do sistema deve ser construída em React Native. |
| **RNF 10** | Escalabilidade | Desempenho | O sistema deve ser escalável, permitindo o aumento da capacidade de processamento e armazenamento conforme a demanda de usuários cresce. |
| **RNF 11** | Backup e recuperação | Confiabilidade | O sistema deve permitir a realização de backups automáticos e recuperação rápida de dados, garantindo a continuidade do serviço em caso de falhas. |
| **RNF 12** | Compatibilidade com APIs | Portabilidade | O sistema deve ser compatível com APIs de terceiros, garantindo integração com outros sistemas e fontes de dados. |
| **RNF 13** | Log de eventos | Segurança | O sistema deve registrar logs detalhados de todas as operações críticas e tentativas de acesso, para facilitar a auditoria e rastreamento de falhas. |
| **RNF 14** | Tempo de resposta | Performance | O sistema deve responder a solicitações do usuário em tempo inferior a 2 segundos para garantir uma experiência rápida e eficiente. |
| **RNF 15** | Acessibilidade | Usabilidade | O sistema deve atender aos padrões de acessibilidade, permitindo que usuários com deficiência visual ou auditiva possam utilizá-lo de forma eficaz. |
| **RNF 16** | Internacionalização e localidade | Portabilidade | O sistema deve permitir tradução e adaptação para diferentes idiomas e fusos horários, tornando-se acessível para um público global. |
| **RNF 17** | Automação de testes | Manutenibilidade | O sistema deve suportar a automação de testes para garantir que as funcionalidades sejam testadas de forma eficiente e com alta cobertura. |
| **RNF 18** | Segurança de rede | Segurança | O sistema deve utilizar criptografia de ponta a ponta para proteger os dados em trânsito, garantindo a segurança nas comunicações de rede. |
| **RNF 19** | Consumo de recursos | Performance | O sistema deve ser otimizado para consumir recursos (como CPU e memória) de forma eficiente, garantindo um baixo impacto no desempenho do dispositivo. |
| **RNF 20** | Redundância | Confiabilidade | O sistema deve possuir redundância de hardware e de rede, garantindo alta disponibilidade e minimizando os riscos de falhas. |

Fonte: Acervo do autor.

5.2. API da BASE DE DADOS

A API da base de dados é responsável por receber os dados da API de consulta ao broker e de regra de negócios e efetuar as operações de CRUD ( Create, Read, Update e Delete) na base de dados PostgreSQL.

5.2.1 Requisitos funcionais

No quadro xx pode-se visualizar os requisitos funcionais da API de manipulação da base de dados.

**Quadro 12 – Requisitos funcionais**

|  |  |  |  |
| --- | --- | --- | --- |
| **Nº** | **Nome** | **Classificação** | **Descrição** |
| **RF 01** | Login do sistema | Segurança | O sistema deverá dispor meios de autenticação para usuários e administradores. |
| **RF 02** | Cadastro, alteração e exclusão de usuários | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de usuários. |
| **RF 03** | Cadastro, alteração e exclusão de produtos | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de produtos. |
| **RF 04** | Cadastro, alteração e exclusão de categorias | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de categorias. |
| **RF 05** | Cadastro, alteração e exclusão de subcategorias | Usabilidade | O sistema deve possibilitar o cadastro de subcategorias de um produto. |
| **RF 06** | Cadastro, alteração e exclusão de estoque | Usabilidade | O sistema deve permitir o cadastro, alteração e exclusão de itens no estoque, atualizando automaticamente as quantidades disponíveis. |
| **RF 07** | Controle de permissões | Segurança | O sistema deverá dispor de um controle de permissões para definir os níveis de acesso de usuários (ex.: administrador, operador). |
| **RF 08** | Exibição de itens | Usabilidade | O sistema deverá dispor de uma rotina que exiba todos os itens cadastrados. |
| **RF 09** | Quantidade | Usabilidade | O sistema deverá dispor de uma rotina que exiba a quantidade em Kg de cada item cadastrado. |
| **RF 10** | Setar quantidade mínima | Usabilidade | O sistema deverá dispor de uma rotina que permita setar a quantidade mínima de um produto. |
| **RF 11** | Alertas | Confiabilidade | O sistema deverá dispor de uma rotina que emita alertas de acordo com a quantidade mínima configurada. |
| **RF 12** | Entrada e saída de itens | Rastreabilidade | O sistema deverá dispor de uma rotina que registre a data de quando um item é adicionado e quando é retirado. |
| **RF 13** | Número de refeições | Usabilidade | O sistema deverá dispor de uma rotina que calcule o número de refeições restantes que cada item possui. |
| **RF 14** | Segurança de senhas | Segurança | O sistema deverá exigir senhas com no mínimo 8 caracteres, incluindo letras maiúsculas e números. |
| **RF 15** | Armazenamento de senhas | Segurança | O sistema deverá gravar as senhas de forma criptografada no banco de dados. |

Fonte: Acervo do autor.

5.2.2 Requisitos não funcionais

No quadro xx pode-se visualizar os requisitos não funcionais da API de manipulação da base de dados.

**Quadro 11 – Requisitos não funcionais**

|  |  |  |  |
| --- | --- | --- | --- |
| **Nº** | **Nome** | **Classificação** | **Descrição** |
| **RNF 01** | Multiplataforma | Portabilidade | O sistema deve ser multiplataforma, permitindo ao usuário maior flexibilidade de acesso e produtividade. |
| **RNF 02** | Consistência de dados | Confiabilidade | O sistema deve ser desenvolvido para que as informações inseridas e consultadas sejam consistentes. |
| **RNF 03** | Segurança do sistema | Segurança | O sistema deve adotar política e medidas de segurança. |
| **RNF 04** | Interface amigável | Usabilidade | O sistema deve ser desenvolvido de forma que o usuário tenha facilidade em sua utilização. |
| **RNF 05** | Consulta e gravação de dados | Performance | O sistema deve possuir boa performance para consulta e gravação no banco de dados. |
| **RNF 06** | Manutenção | Manutenibilidade | O sistema deve ser de fácil manutenibilidade, usando padrão de projetos para permitir fácil implementação de novas funcionalidades e correção de erros. |
| **RNF 07** | Servidor | Confiabilidade | O sistema deve ser hospedado em servidor cloud e ter backups diários da base de dados. |
| **RNF 08** | Programação backend | Padrões | A programação backend do sistema deve ser construída em C#. |
| **RNF 09** | Programação frontend | Padrões | A programação frontend do sistema deve ser construída em React Native. |
| **RNF 10** | Escalabilidade | Desempenho | O sistema deve ser escalável, permitindo o aumento da capacidade de processamento e armazenamento conforme a demanda de usuários cresce. |
| **RNF 11** | Backup e recuperação | Confiabilidade | O sistema deve permitir a realização de backups automáticos e recuperação rápida de dados, garantindo a continuidade do serviço em caso de falhas. |
| **RNF 12** | Compatibilidade com APIs | Portabilidade | O sistema deve ser compatível com APIs de terceiros, garantindo integração com outros sistemas e fontes de dados. |
| **RNF 13** | Log de eventos | Segurança | O sistema deve registrar logs detalhados de todas as operações críticas e tentativas de acesso, para facilitar a auditoria e rastreamento de falhas. |
| **RNF 14** | Tempo de resposta | Performance | O sistema deve responder a solicitações do usuário em tempo inferior a 2 segundos para garantir uma experiência rápida e eficiente. |
| **RNF 15** | Acessibilidade | Usabilidade | O sistema deve atender aos padrões de acessibilidade, permitindo que usuários com deficiência visual ou auditiva possam utilizá-lo de forma eficaz. |
| **RNF 16** | Internacionalização e localidade | Portabilidade | O sistema deve permitir tradução e adaptação para diferentes idiomas e fusos horários, tornando-se acessível para um público global. |
| **RNF 17** | Automação de testes | Manutenibilidade | O sistema deve suportar a automação de testes para garantir que as funcionalidades sejam testadas de forma eficiente e com alta cobertura. |
| **RNF 18** | Segurança de rede | Segurança | O sistema deve utilizar criptografia de ponta a ponta para proteger os dados em trânsito, garantindo a segurança nas comunicações de rede. |
| **RNF 19** | Consumo de recursos | Performance | O sistema deve ser otimizado para consumir recursos (como CPU e memória) de forma eficiente, garantindo um baixo impacto no desempenho do dispositivo. |
| **RNF 20** | Redundância | Confiabilidade | O sistema deve possuir redundância de hardware e de rede, garantindo alta disponibilidade e minimizando os riscos de falhas. |

Fonte: Acervo do autor.

**Quadro 12 – Requisitos funcionais**

|  |  |  |  |
| --- | --- | --- | --- |
| **Nº** | **Nome** | **Classificação** | **Descrição** |
| **RF 01** | Login do sistema | Segurança | O sistema deverá dispor meios de autenticação para usuários e administradores. |
| **RF 02** | Cadastro, alteração e exclusão de usuários | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de usuários. |
| **RF 03** | Cadastro, alteração e exclusão de produtos | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de produtos. |
| **RF 04** | Cadastro, alteração e exclusão de categorias | Usabilidade | O sistema deve possibilitar o cadastro, alteração e exclusão de categorias. |
| **RF 05** | Cadastro, alteração e exclusão de subcategorias | Usabilidade | O sistema deve possibilitar o cadastro de subcategorias de um produto. |
| **RF 06** | Cadastro, alteração e exclusão de estoque | Usabilidade | O sistema deve permitir o cadastro, alteração e exclusão de itens no estoque, atualizando automaticamente as quantidades disponíveis. |
| **RF 07** | Controle de permissões | Segurança | O sistema deverá dispor de um controle de permissões para definir os níveis de acesso de usuários (ex.: administrador, operador). |
| **RF 08** | Exibição de itens | Usabilidade | O sistema deverá dispor de uma rotina que exiba todos os itens cadastrados. |
| **RF 09** | Quantidade | Usabilidade | O sistema deverá dispor de uma rotina que exiba a quantidade em Kg de cada item cadastrado. |
| **RF 10** | Setar quantidade mínima | Usabilidade | O sistema deverá dispor de uma rotina que permita setar a quantidade mínima de um produto. |
| **RF 11** | Alertas | Confiabilidade | O sistema deverá dispor de uma rotina que emita alertas de acordo com a quantidade mínima configurada. |
| **RF 12** | Entrada e saída de itens | Rastreabilidade | O sistema deverá dispor de uma rotina que registre a data de quando um item é adicionado e quando é retirado. |
| **RF 13** | Número de refeições | Usabilidade | O sistema deverá dispor de uma rotina que calcule o número de refeições restantes que cada item possui. |
| **RF 14** | Segurança de senhas | Segurança | O sistema deverá exigir senhas com no mínimo 8 caracteres, incluindo letras maiúsculas e números. |
| **RF 15** | Armazenamento de senhas | Segurança | O sistema deverá gravar as senhas de forma criptografada no banco de dados. |

Fonte: Acervo do autor.

5.3. MATRIZ DE REQUISITOS

Este capítulo traz informações sobre as matrizes de requisitos e suas dependências.

6. Balança Inteligente com Sensor de Peso e Arduino Nano ESP32

Neste capítulo será abordada a construção da balança inteligente, desenhos esquemas e materiais.

6.1. Lista de materiais

A tabela abaixo contém a lista de materiais utilizados para a construção física do protótipo.

**Quadro 11 – Lista de componentes**

|  |  |
| --- | --- |
| Componente | Características |
| Sensor de Peso | Ponte completa, não precisa trabalhar em conjunto com outros sensores de carga.  Furos de fixação com rosca M4 e M5. |
| Módulo Conversor HX711 para Sensor de Peso | Módulo conversor e amplificador HX711 de 24 bits, utilizado para amplificar o sinal de dispositivos como células de carga, fazendo a interligação entre essas células e o microcontrolador. |
| Protoboard 400 Pontos | Facilidade de prototipagem.  Tamanho compacto. |
| Arduino Nano ESP32 | Combina as dimensões compactas com o microcontrolador ESP32, fornecendo maior potencial e recursos de WiFi e Bluetooth para o projeto. |
| Jumpers Macho-Fêmea | Jumpers de conexão tipo Macho-Fêmea. Excelente para montagem de projetos em protoboard com rapidez, agilidade e limpeza. |
| Cabo USB Tipo C | Transmite energia e dados, ideal para o uso em equipamentos com porta USB tipo C e se conecta facilmente em notebooks e placas arduíno. |

Fonte: Elaborado a partir de (Guse, 2024)

6.2. PROGRAMAÇÃO ESP32

7. programação backend

8. programação frontend

9. RESULTADOS DO TC (recomenda-se usar o título do tc aqui)

10. CONCLUSÃO

**REFERÊNCIAS**

ALBAHARI, Joseph. **C# 9.0 in a Nutshell:** The Definitive Reference. O’Reilly Media, Inc., 2021. *Ebook.*

CARVALHO, Vinícius. **PostgreSQL:** Banco de dados para aplicações web modernas. Casa do Código, 2017. *Ebook.*

DABIT, Nader. **React Native in Action:** developing ios and android apps with javascript. Shelter Island: Manning Publications Co., 2019.

EISENMAN, Bonnie. **Learning React Native**. Gravenstein Highway North: O’Reilly Media, Inc., 2016.

FERRARI, Luca; PIROZZI, Enrico. **Learn PostgreSQL.** Birmingham, UK: Packt Publishing Ltd., 2020.

CARRION, Patrícia; QUARESMA, Manuela. **Internet da Coisas (IoT):** Definições e aplicabilidade aos usuários finais. Florianópolis: Programa de Pós-Graduação em Design da Universidade do Estado de Santa Catarina, 27 mar. 2019. Semestral. Disponível em: <https://www.revistas.udesc.br/index.php/hfd/issue/view/662>. Acesso em: 16 jun. 2024.

HAGINO, Taiji Practical. **Node-RED Programming.** Birmingham, UK:Packt Publishing Ltd, 2021.

IDEALI, Wagner. **Conectividade em Automação e IoT**: Protocolos I2C, SPI, USB, TCP-IP entre outros. Funcionalidade e interligação para automação e IoT**.** Editora Alta Books, 2021. *E-book.*

INSOMMIA. **Documentação Oficial.** Disponível em: < https://docs.insomnia.rest/insomnia/get-started>. Acesso em: 10 jul. 2024.

LEDUR, Cleverson Lopes. **Desenvolvimento de Sistemas com C#.** Porto Alegre: Sagah, 2018.

MASCHIETTO et al. **Arquitetura e Infraestrutura de IoT.** Grupo A, 2021. *E-book.*

RAMGIR, Mayur. **Internet of Things**: **Architecture, Implementation and Security.** Noida: Pearson Education India, 2019. *E-book.*

Microsoft Corp. **Documentação sobre os conceitos básicos do .NET.** Disponível em: <https://learn.microsoft.com/pt-br/dotnet/core/introduction>. Acesso em: 08 jun. 2024.

Eclipse Mosquitto, An open source MQTT broker. **Documentação oficial.** Disponível em: < https://mosquitto.org/>. Acesso em: 16 jul. 2024.

ONER, Vedat Ozan. **Developing IoT Projects with ESP32.** Birmingham, UK:Packt Publishing Ltd., 2021.

Rodenburg, J. (2021). **Code like a Pro in C#.** Shelter Island: Manning Publications Co.

The PostgreSQL Global Development Group. **PostgreSQL 16.3 Documentation.** Disponível

em: < https://www.postgresql.org/docs/current>. Acesso em: 08 jun. 2024.

SANCTIS, Valerio De. **Building Web APIs with ASP.NET Core.** Shelter Island:Manning

Publications Co., 2023. *E-book.*SINCLAIR, Bruce. **IoT:** como usar a internet das coisas para alavancar seus negócios. Grupo Autêntica, 2018. *E-book.*

Steve’s Internet Guide. **Mosquitto: Documentação hospedada por terceiros.** Disponível

em: < http://www.steves-internet-guide.com/mqtt/>. Acesso em: 19 jul. 2024.

TROELSEN, Andrew; JAPIKSE, Phil. **Pro C# 10 with .NET 6:** Foundational Principles and Practices in Programming.Chambersburg: Apress Berkeley, CA, 2022. *E-book.*

**ANEXOS**