

Relatório Final

Sicoin - Sistema de Otimização de Coleta de Resíduos em Cidades Inteligentes

João Gabriel Tavares Felix Monteiro
João Victor Rosa Couto e Silva
Karlla Loane Santos Lima
Murilo Henrique de Sousa Freua

Goiânia
2024

Sumário

1 Introdução	2
1.1 Contextualização e Objetivos Gerais	2
1.2 Motivação e Relevância do Projeto	2
1.3 Alinhamento com ODS	2
2 Descrição do Projeto	3
2.1 Escopo do Sistema	3
2.1.1 Funcionalidades do Sistema	3
2.1.2 Limitações do Sistema	4
2.2 Visão Geral da Arquitetura	4
2.3 Tecnologias Utilizadas	5
2.4 Papel da Equipe	7
3 Conceitos Relacionados ao Projeto	8
3.1 Ciência de Contexto	8
3.2 Processamento de Streams e CEP (Complex Event Processing)	9
3.3 Gêmeo Digital	9
3.4 Offloading Estático	10
3.5 Interface de Usuário Ubíqua	10
4 Desenvolvimento do Sistema	13
4.1 Etapas de Desenvolvimento	13
4.2 Módulos do Sistema	14
4.3 Simulação com Node-RED	16
5 Funcionamento do Sistema	18
5.1 Aplicativo Móvel	18
5.2 Sistema de Gestão e Relatórios	23
6 Conclusão e Trabalhos Futuros	24
A História de Usuários e Cenários	27
B Links úteis	31

1 Introdução

1.1 Contextualização e Objetivos Gerais

O crescimento populacional e o aumento da urbanização têm desafiado a gerencia de recursos de forma eficiente e sustentável. Nesse contexto, a coleta de resíduos é uma das operações urbanas mais críticas, impactando diretamente a limpeza, a saúde pública e o meio ambiente.

Este projeto propõe um sistema de coleta de resíduos em cidades inteligentes, que utiliza tecnologias de computação ubíqua, como sensores IoT, geolocalização e algoritmos de otimização, para monitorar em tempo real o volume de resíduos e gerar rotas dinâmicas para os agentes de coleta. O objetivo geral é otimizar os processos de coleta, promovendo eficiência operacional, economia de recursos e redução das emissões de carbono.

1.2 Motivação e Relevância do Projeto

Os métodos tradicionais de coleta de resíduos muitas vezes resultam em rotas fixas e ineficientes, levando a desperdício de tempo, combustível e recursos financeiros. Além disso, a falta de monitoramento em tempo real dificulta a priorização de áreas críticas e aumenta os impactos ambientais.

Portanto, a relevância do projeto está em sua capacidade de integrar tecnologias avançadas, como sensores IoT e processamento em tempo real, para abordar essas deficiências. Ao implementar um sistema que se adapta às condições dinâmicas das cidades, o projeto não apenas melhora a eficiência operacional, mas também contribui para o desenvolvimento sustentável e a qualidade de vida dos cidadãos.

1.3 Alinhamento com ODS

No contexto de uma cidade inteligente, esse sistema se integra ao ecossistema urbano ao possibilitar a coleta em tempo real de dados de resíduos, alinhando-se com os Objetivos de Desenvolvimento Sustentável (ODS):

- ODS 11 - Cidades e Comunidades Sustentáveis: contribui para o desenvolvimento de cidades mais organizadas e eficientes, reduzindo desperdícios e melhorando a limpeza urbana.
- ODS 13 - Ação Contra a Mudança Global do Clima: contribui na redução das emissões de carbono ao otimizar rotas e minimizar deslocamentos desnecessários.

2 Descrição do Projeto

2.1 Escopo do Sistema

O sistema de coleta de resíduos em cidades inteligentes foi desenvolvido para otimizar o processo de coleta com base no volume de resíduos e na geolocalização dos agentes de limpeza. A solução utiliza sensores instalados em lixeiras para monitorar, em tempo real, o nível de preenchimento de resíduos, enviando essas informações para uma camada de processamento centralizada. Além disso, os dispositivos móveis dos agentes de coleta transmitem continuamente sua localização ao sistema, permitindo o cálculo de rotas dinâmicas e otimizadas.

Com base nesses dados, o sistema sugere trajetos que evitam deslocamentos desnecessários, promovendo economia de recursos e redução de emissões de carbono. Integrado ao ecossistema de uma cidade inteligente, o sistema se adapta às condições dinâmicas do ambiente urbano, oferecendo uma abordagem flexível e orientada por dados. Os agentes de coleta utilizam dispositivos móveis para acessar as rotas atualizadas em tempo real, enquanto gestores monitoram as operações e analisam tendências por meio de uma interface web.

2.1.1 Funcionalidades do Sistema

O sistema desenvolvido incorpora funcionalidades que visam atender às necessidades de um ambiente urbano inteligente, promovendo eficiência e sustentabilidade na coleta de resíduos. Abaixo estão descritas as principais funcionalidades:

1. Monitoramento em tempo real

Sensores IoT simulados monitoram continuamente o nível de resíduos nas lixeiras e enviam os dados para o sistema central. Esses dados permitem identificar quais lixeiras estão próximas de sua capacidade máxima, possibilitando ações rápidas e precisas.

2. Geração automática de rotas otimizadas

Com base nos níveis de preenchimento das lixeiras e na localização dos agentes de coleta, o sistema calcula rotas otimizadas e as envia automaticamente para os dispositivos móveis dos agentes. Essa funcionalidade minimiza deslocamentos desnecessários, economizando tempo e recursos.

3. Notificações em Tempo Real

Durante a coleta, o sistema monitora as condições das lixeiras em tempo real. Caso uma nova lixeira atinja sua capacidade máxima, o sistema recalcula a rota e notifica os agentes, garantindo que demandas emergentes sejam atendidas prontamente.

4. Ajuste em tempo real de rotas com base em mudanças contextuais

O sistema ajusta as rotas de coleta dinamicamente, levando em conta alterações no contexto, como novas lixeiras atingindo capacidade máxima ou mudanças na localização dos agentes. Essas atualizações são enviadas em tempo real aos dispositivos móveis, garantindo maior eficiência e adaptabilidade à operação.

5. Relatórios de eficiência

Supervisores têm acesso a relatórios que analisam o desempenho das coletas realizadas. Os relatórios incluem métricas como distância percorrida, tempo gasto, emissões de carbono, permitindo uma avaliação precisa da eficiência operacional.

Para mais detalhes sobre as funcionalidades e seus cenários, consulte o Anexo **A - Histórias de Usuário e Cenários**.

2.1.2 Limitações do Sistema

É importante ressaltar que o sistema desenvolvido é um protótipo funcional e apresenta algumas limitações, a saber:

- Não utiliza sensores físicos reais; os dados são simulados para testes e validação;
- Não abrange a destinação final dos resíduos, limitando-se à coleta e monitoramento;
- Depende de conectividade constante para realizar operações em tempo real.

2.2 Visão Geral da Arquitetura

A arquitetura do sistema, representada na figura **I**, foi projetada para integrar dispositivos IoT, processamento em nuvem e interfaces de usuário móveis e web, garantindo comunicação em tempo real e escalabilidade. A camada IoT, simulada pelo Node-RED, envia dados sobre o estado das lixeiras para o broker MQTT, hospedado na AWS, que gerencia a troca de mensagens entre dispositivos e o servidor de aplicação. O servidor, também hospedado na AWS, é responsável por processar esses dados, manter o gêmeo digital das lixeiras atualizado e interagir com a API do Google Maps para gerar rotas otimizadas. Essas informações são então transmitidas aos dispositivos móveis e ao portal web via APIs REST e WebSocket.

Além disso, a arquitetura inclui a utilização de Docker e Docker Compose para orquestração de containers, Mosquitto como broker MQTT, Keycloak para autenticação e autorização via OAuth 2.0, e instâncias AWS EC2 para hospedagem. A integração de todos esses componentes permite uma operação eficiente e adaptável ao contexto. Mais detalhes sobre a arquitetura e sua implementação podem ser encontrados no Anexo **B - Links Úteis**.

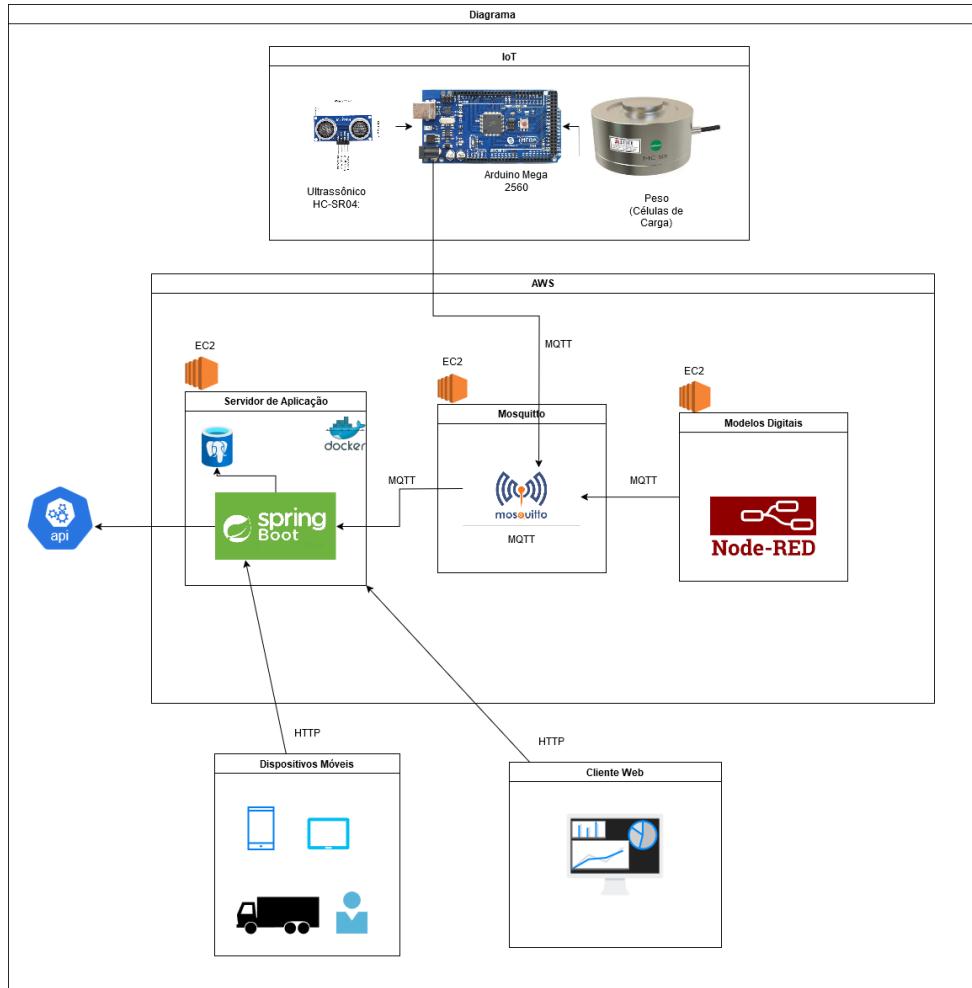


Figura 1: Visão geral da arquitetura e implantação do sistema

2.3 Tecnologias Utilizadas

Nesta seção, apresentamos as tecnologias empregadas no desenvolvimento deste projeto, acompanhadas das justificativas para suas escolhas. A utilização detalhada de cada tecnologia e sua integração no sistema será explorada na Seção 4 - Desenvolvimento do Sistema, onde abordaremos as implementações específicas e a interação entre os componentes.

2.3.1 Node-RED

O Node-RED é uma ferramenta de código aberto que foi utilizada para simular o comportamento dos sensores IoT no projeto.

Devido à complexidade de aquisição e implementação da conexão de sensores físicos reais no escopo do projeto, o Node-RED foi escolhido por sua facilidade em simular fluxos de dados em tempo real. Ele permite emular as condições reais do sistema, como mudanças no nível de resíduos das lixeiras, e facilita a integração com o backend via MQTT.

2.3.2 MQTT (Message Queuing Telemetry Transport)

O MQTT, protocolo leve de comunicação, foi usado para transmitir dados dos sensores IoT simulados para o backend do sistema.

Sua utilização neste projeto se justifica pela eficiência em redes de baixa largura de banda e pela capacidade de lidar com dispositivos IoT que enviam dados constantemente. No projeto, ele garante a entrega de mensagens em tempo real, essencial para detectar rapidamente mudanças no nível de resíduos das lixeiras. Além disso, seu suporte a Qualidade de Serviço (QoS) oferece confiabilidade para garantir a entrega dos dados, mesmo em condições de conectividade instáveis.

2.3.3 Java + Spring boot

Para o desenvolvimento do back-end foi utilizada a linguagem Java e o framework Spring Boot, sendo responsável pelo processamento dos dados recebidos, geração de rotas e disponibilização de APIs RESTful.

A combinação de Java com Spring Boot foi escolhida por sua robustez e flexibilidade na criação de sistemas distribuídos. O backend, sendo o núcleo do sistema, exige uma arquitetura escalável e modular para processar os dados dos sensores em tempo real e disponibilizar rotas otimizadas para os agentes. O Spring Boot, além de facilitar o desenvolvimento rápido de APIs, oferece integração eficiente com o PostgreSQL e ferramentas de segurança, garantindo a confiabilidade e a escalabilidade necessárias ao projeto.

2.3.4 PostgreSQL

O PostgreSQL é um banco de dados relacional que neste projeto foi utilizado para armazenar os dados históricos de coleta, relatórios gerados e informações dos sensores.

Ele foi escolhido por sua capacidade de gerenciar dados complexos de forma eficiente e suportar consultas analíticas, necessárias para gerar relatórios de eficiência e tendências de geração de resíduos.

2.3.5 Flutter

Framework utilizado para o desenvolvimento do aplicativo móvel destinado aos agentes de coleta.

O Flutter foi escolhido por sua capacidade de desenvolver aplicativos multiplataforma, reduzindo o esforço e o tempo de desenvolvimento. Além disso, sua compatibilidade com SQLite permite armazenar dados offline, essencial para o caso de indisponibilidade de conexão durante o turno de coleta.

2.3.6 React

React é uma biblioteca JavaScript para construção de interfaces de usuário e foi utilizada para o desenvolvimento do portal web gerencial, acessado pelos supervisores. Sua escolha se deu pela eficiência em criar interfaces dinâmicas e interativas, que facilitam a visualização de dashboards e relatórios em tempo real. Isso é fundamental para supervisores monitorarem as operações e tomarem decisões estratégicas com base nos dados apresentados.

2.3.7 API do Google Maps

Foi utilizada a API do Google Maps para cálculo e exibição de rotas otimizadas, e também para fornecer visualizações geográficas para os agentes e supervisores.

A API do Google Maps foi escolhida pela confiabilidade e precisão na geração de rotas, elementos essenciais para um sistema que depende de otimização em tempo real para reduzir deslocamentos desnecessários.

2.3.8 AWS

A AWS, plataforma de serviços de computação em nuvem, foi utilizada para hospedar o backend e os serviços do sistema.

A AWS foi escolhida por oferecer serviços escaláveis e confiáveis, essenciais para lidar com o processamento em tempo real dos dados dos sensores IoT e a geração dinâmica de rotas. Além disso, sua infraestrutura garante alta disponibilidade para supervisores e agentes de coleta.

2.4 Papel da Equipe

A equipe de desenvolvimento é composta por quatro integrantes, com responsabilidades distribuídas conforme as habilidades individuais de cada membro, visando o progresso eficiente do projeto. A tabela a seguir apresenta a composição da equipe e os papéis desempenhados por cada integrante.

Integrante	Papel
João Gabriel	<ul style="list-style-type: none"> - Desenvolvimento do back-end com ênfase em processamento dos dados e integração com API do Google Maps; - Banco de dados; - Implantação.
João Vitor	<ul style="list-style-type: none"> - Desenvolvimento do portal web para supervisores.
Karlla	<ul style="list-style-type: none"> - Desenvolvimento do aplicativo móvel; - Documentação de requisitos do sistema e documentação do projeto.
Murilo	<ul style="list-style-type: none"> - Simulação dos sensores IoT no Node-RED; - Desenvolvimento do back-end;

Tabela 1: Papéis e responsabilidades

3 Conceitos Relacionados ao Projeto

3.1 Ciência de Contexto

A ciência de contexto, no âmbito da computação ubíqua, refere-se à capacidade de sistemas computacionais de coletar, interpretar e reagir a informações contextuais para oferecer serviços adaptativos e relevantes. Contexto é definido como qualquer informação que pode ser usada para caracterizar a situação de uma entidade, como uma pessoa, lugar ou objeto relevante para a interação entre usuário e aplicação.

No projeto desenvolvido, a ciência de contexto é aplicada ao coletar e processar dados de sensores IoT, como os níveis de resíduos em lixeiras, e de dispositivos móveis, como a localização dos agentes de coleta. Essas informações contextuais são analisadas em tempo real para inferir o estado atual do ambiente. Com base nessa análise, o sistema calcula rotas otimizadas e as ajusta dinamicamente sempre que ocorrerem mudanças, como uma lixeira atingindo sua capacidade máxima ou alterações na posição dos agentes.

Assim, essa abordagem permite que o sistema tome decisões baseadas no estado atual do ambiente, minimizando deslocamentos desnecessários e priorizando a eficiência e a economia de recursos.

Importante mencionar que o aplicativo também se adapta ao contexto do dispositivo, permitindo uma completa visualização dos dados, mesmo em dispositivos de tamanhos diferentes, ou por meio de mudanças na rotação do dispositivo.

3.2 Processamento de Streams e CEP (Complex Event Processing)

O processamento de streams e o Complex Event Processing (CEP) são técnicas fundamentais em sistemas que lidam com grandes volumes de dados em tempo real. O processamento de streams permite analisar fluxos contínuos de dados, enquanto o CEP identifica padrões ou eventos complexos a partir de múltiplos dados simples, gerando informações relevantes [4].

No sistema desenvolvido, essas técnicas são aplicadas em três contextos principais:

1. Monitoramento simples da capacidade das lixeiras

O sistema processa continuamente os dados de preenchimento das lixeiras. Quando uma lixeira ultrapassa um limiar específico (por exemplo, 40%), ela é marcada como candidata para a rota de coleta.

2. Identificação de novas lixeiras durante a rota

Durante a execução de uma rota, o sistema monitora em tempo real se lixeiras próximas à rota de coleta atingem um valor mais elevado, como 70% de sua capacidade. Quando detectado, o CEP infere que uma nova lixeira precisa ser incluída naquela rota, acionando o recálculo dinâmico para otimizar a coleta.

3. Detecção de eventos de coleta

O sistema analisa eventos mais complexos, como a combinação de geolocalização do caminhão e mudanças abruptas nos níveis de resíduos. Por exemplo, se uma lixeira próxima ao caminhão tem seu volume reduzido drasticamente, o CEP infere que a lixeira foi coletada e registra essa informação automaticamente, sem intervenção manual.

3.3 Gêmeo Digital

O gêmeo digital é definido como um conjunto de construções informacionais que representa um sistema físico de forma conectada, durante todo o seu ciclo de vida, por meio de um processo contínuo de captura, processamento e realimentação de dados. Essa conexão permite que o gêmeo digital reflita o estado atual do sistema físico, fornecendo informações detalhadas sobre seu gêmeo físico.

No sistema desenvolvido, cada lixeira monitorada possui um gêmeo digital de instância, que reflete o estado desta durante todo o ciclo de vida, permitindo monitoramento e auditoria. Este gêmeo digital é mantido por meio de dados capturados em tempo real pelos sensores IoT, incluindo informações como o nível de preenchimento e peso, localização geográfica e condições recentes (por exemplo, se foi coletada).

Importante ressaltar que o objetivo deste trabalho não foi construir um gêmeo digital completo do ambiente para simulação avançada de cenários, mas sim prover um modelo funcional

de consulta e monitoramento. Este modelo reflete o estado real e atual das lixeiras físicas, garantindo que decisões baseadas no contexto possam ser tomadas com precisão e confiabilidade.

3.4 Offloading Estático

O offloading refere-se à transferência de partes do processamento de dados de um dispositivo com recursos limitados, como um smartphone, para ambientes mais robustos, como servidores na nuvem ou infraestrutura local. No sistema desenvolvido, foi adotado o modelo de offloading estático [3], em que as tarefas e responsabilidades de processamento são previamente definidas.

No projeto, o aplicativo móvel dos agentes é responsável por tarefas leves, como exibir rotas e enviar a localização em tempo real, enquanto o backend, hospedado na nuvem, realiza o processamento intensivo, como análise dos dados dos sensores, geração de rotas otimizadas e detecção de eventos contextuais. Essa estratégia reduz o consumo de recursos dos dispositivos móveis, garantindo maior eficiência e responsividade, mesmo em cenários com limitações de hardware ou energia.

3.5 Interface de Usuário Ubíqua

Na computação ubíqua, uma interface de usuário eficaz deve atender aos 3 Cs: consistência, continuidade e complementariedade. Esses princípios garantem que a experiência do usuário seja adaptativa, fluida e eficiente, independentemente do dispositivo ou contexto de uso. Discutiremos sobre a aplicação destes princípios no trabalho.

3.5.1 Consistência

Consistência refere-se à uniformidade na experiência do usuário entre diferentes dispositivos. O aplicativo foi desenvolvido para manter a mesma experiência de usuário entre diferentes dispositivos, como smartphones e tablets. O conteúdo, o fluxo de navegação, as estruturas e as funcionalidades são replicados, garantindo que o usuário possa alternar entre dispositivos sem a necessidade de reaprendizagem.

Abaixo, vemos um exemplo de fluxo para iniciar coleta em um smartphone (figura 2) e em um tablet (figura 3), onde aplicativo exibe informações de rotas e mapa de forma uniforme, independentemente do dispositivo utilizado. Além disso, é possível observar a consistência no design, incluindo a aplicação do modo noturno, que mantém a mesma experiência visual em ambos os dispositivos.

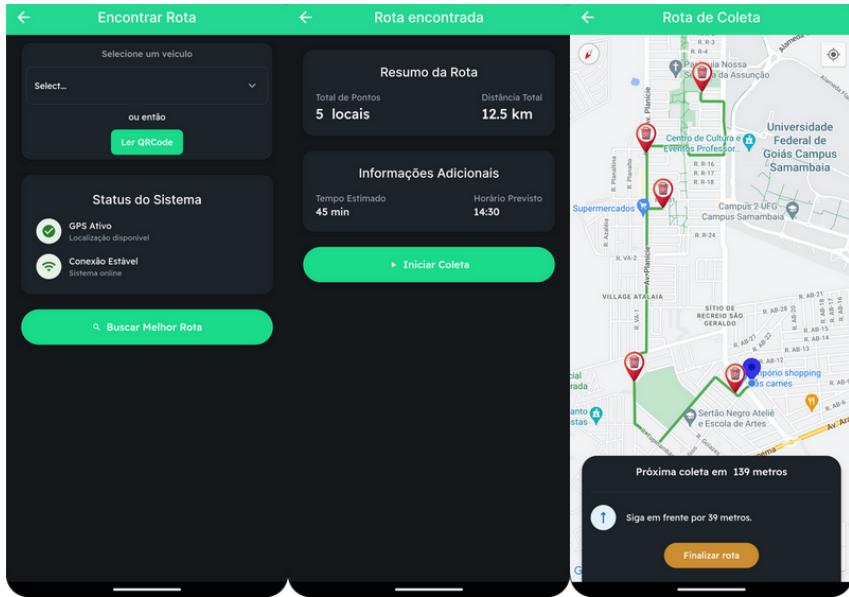


Figura 2: Fluxo mobile



Figura 3: Fluxo tablet

3.5.2 Continuidade

A continuidade visa proporcionar uma transição fluida entre dispositivos, permitindo que o usuário continue uma atividade sem interrupções ou reinício. Neste sistema ela foi desenvolvida para garantir que o estado da aplicação seja sincronizado entre dispositivos conectados.

Por exemplo, se o agente está com uma rota aberta em um dispositivo e realiza login em outro, o sistema sincroniza o estado e abre diretamente a página de navegação no segundo dispositivo, em vez de abrir para a página de "Encontrar rota". Da mesma forma, se a rota é finalizada em um dispositivo, essa ação é refletida imediatamente no outro, retornando ambos à página de "Encontrar Rota".

Essa funcionalidade é especialmente útil em cenários em que a bateria do dispositivo principal está acabando ou já se esgotou. O agente pode simplesmente alternar para outro dispositivo e continuar seu trabalho sem perder a sessão ou progresso.

Embora a continuidade dependa de conexão com a internet para funcionar, essa restrição está alinhada à arquitetura geral do sistema, que já requer conectividade para operações como

sincronização de rotas e notificações em tempo real.

Um vídeo demonstrando esta funcionalidade pode ser acessado em: [Link](#).

3.5.3 Complementariedade

A complementariedade diz respeito aos dispositivos colaborem entre si, seja por meio de informações complementares ou funcionalidades que se adaptam ao contexto e às capacidades de cada dispositivo.

A interface do aplicativo foi desenvolvida considerando a responsividade das telas, adaptando sua apresentação às características do dispositivo em uso, complementando a experiência do usuário de acordo com o contexto:

- Em smartphones com orientação vertical, exibe apenas o mapa e instruções de navegação para melhor visualização.
- Em tablets ou smartphones na orientação horizontal, aproveita o espaço adicional para incluir funcionalidades complementares, como uma lista lateral das lixeiras na rota, indicando as já visitadas e as restantes.

Essa abordagem responsiva permite que a interface se ajuste ao dispositivo utilizado, otimizando a interação e complementando a experiência do usuário. Na figura 4 abaixo vemos o exemplo de complementariedade das informações no aplicativo, onde do lado esquerdo temos a interface para smartphones e do lado esquerdo a interface para tablets.

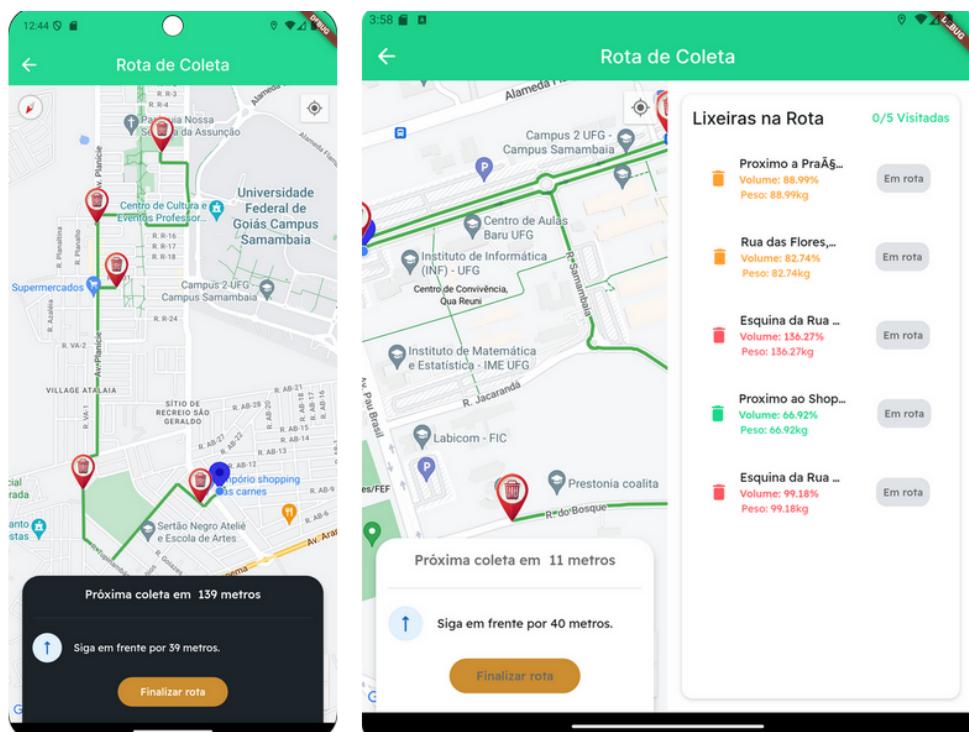


Figura 4: Complementariedade de informações

4 Desenvolvimento do Sistema

4.1 Etapas de Desenvolvimento

O desenvolvimento do trabalho foi realizado nas seguintes etapas:

1. Definição do escopo do projeto

2. Levantamento de requisitos

- Definição das necessidades do sistema;
- Elaboração de histórias de usuário e cenários;
- Priorização das funcionalidades com base no escopo e no prazo.

3. Planejamento e design arquitetural

- Escolha das tecnologias e ferramentas adequadas para o projeto;
- Criação do documento de arquitetura, incluindo definições de camadas, comunicação e visão geral do sistema.

4. Implementação inicial e integração

- Implementação dos fluxos de simulação no Node-RED para simular sensores IoT;
- Configuração inicial dos projetos backend, frontend e mobile.

5. Desenvolvimento

- Backend: implementação de algoritmos de otimização de rotas, integração com banco de dados PostgreSQL e processamento de eventos (CEP);
- Frontend: desenvolvimento do portal web para supervisores com dashboards e gráficos de tendência;
- Aplicativo mobile: criação do aplicativo para agentes de coleta.

6. Validação e ajustes

- Simulações com dados do Node-RED para testar cenários de coleta;
- Refinamento de interfaces para melhorar a usabilidade.

7. Documentação final

- Atualização do documento de arquitetura;
- Elaboração do relatório para a disciplina de computação ubíqua.

4.2 Módulos do Sistema

Nesta seção iremos detalhar os principais módulos que foram desenvolvidos para o funcionamento do protótipo do sistema.

4.2.1 Backend

O backend do sistema foi denominado **SicoIn** (Sistema de Coleta Inteligente). Ele é composto por uma arquitetura modular conteinerizada, que constitui a camada de servidor de aplicação. Os módulos principais são

- sicoIn_server: container que abriga o servidor principal, responsável pela lógica de negócio e de persistência dos dados.
- postgres: coontainer que executa uma instância do banco de dados PostgreSQL, utilizado para armazenamento e consultas de dados.
- Keycloak: container responsável por abrigar o servidor de autenticação, autorização e gestão de usuários, o qual é um software open-source configurável.

O servidor foi implementado em Java 21 com o framework Spring Boot, que oferece suporte a diversos módulos essenciais para a construção de APIs RESTful e integração com o MQTT. Foi utilizada uma abordagem RESTfull para comunicação com as camadas web tradicionais e móvel e MQTT para as camadas IoT. O processamento de dados no backend é orientado por eventos, permitindo reatividade em tempo real.

Casos de Usos principais:

1. Inferência de coleta

O servidor foi projetado para reconhecer automaticamente eventos de coleta em cenários ubíquos, dispensando a necessidade de intervenção manual. Essa lógica utiliza:

- Detecção de esvaziamento de lixeira: O servidor monitora quando uma lixeira reduz abruptamente seu volume de resíduos.
- Confirmação pelo dispositivo móvel: O aplicativo informa ao servidor que o caminhão esteve próximo à lixeira.

Caso ambas as condições sejam atendidas, o servidor registra a coleta como confirmada.

2. Obtenção e descobrimento de rotas em tempo real

- O servidor mantém o estado atualizado das lixeiras como um gêmeo digital, refletindo suas condições em tempo real.

- Ao iniciar a rota, o caminhão recebe uma rota otimizada gerada pela API do Google Maps, intermediada pelo servidor.
- Durante a coleta, o servidor verifica continuamente se novas lixeiras atingiram capacidade crítica e ajusta a rota para incluí-las. Simultaneamente, as lixeiras já coletadas são removidas das próximas rotas. Essa lógica assegura uma operação dinâmica e reativa ao contexto, alinhando-se aos princípios da computação ubíqua.

4.2.2 Frontend - Portal web

O sicoин-frontend é a parte visual referente aos Relatórios do Sistema de Coleta Inteligente, sendo desenvolvido a partir do Next.js, um framework baseado em React que facilita a criação de aplicações web com renderização no lado do servidor (SSR) e geração de páginas estáticas.

Para estilização, o projeto adota o Tailwind CSS, um framework CSS utilitário que permite criar layouts responsivos e personalizáveis de forma prática.

As rotas da aplicação são gerenciadas pelo Next.js e estão organizadas no diretório pages. Além das páginas principais, o projeto inclui rotas de API no diretório pages/api, permitindo que o frontend se comunique com os serviços de backend, como consulta de dados ou envio de informações.

Um ponto importante é a presença do diretório prisma, que indica o uso do Prisma ORM para interagir com um banco de dados. Isso sugere que a aplicação trabalha com dados dinâmicos, como registros de coleta ou informações do sistema, e possibilita operações de leitura, escrita e atualização de forma estruturada.

Para rodar o projeto localmente, basta instalar as dependências com `npm install` e iniciar o servidor de desenvolvimento com `npm run dev`. Com isso, o sistema estará disponível em `http://localhost:3000`. A configuração também inclui um arquivo `.env`, que permite gerenciar variáveis sensíveis, como credenciais ou URLs de serviços externos.

4.2.3 Aplicativo Móvel

O aplicativo mobile, chamado de Coleta+, foi desenvolvido para ser utilizado pelos agentes de coleta como ferramenta principal para acessar as rotas otimizadas e interagir com o sistema em tempo real. Construído com o framework Flutter, o aplicativo oferece compatibilidade com dispositivos Android e iOS, garantindo maior abrangência de uso.

Funcionalidades

O aplicativo foi desenvolvido para atender aos requisitos funcionais do sistema, permitindo que os agentes de coleta visualizem rotas otimizadas com base no nível de preenchimento

das lixeiras e na localização do agente. Além disso, oferece notificações em tempo real sobre mudanças contextuais, como novas lixeiras a serem coletadas, e funcionalidades adaptadas, como o registro do turno. Além disso, o layout do aplicativo é responsivo, exibindo informações adaptadas ao dispositivo e à orientação, incluindo mapas e listas de lixeiras visitadas em tablets ou smartphones na horizontal.

Componente e Widgets Personalizados

O widget personalizado `CustomMapWithRouteAndBins` foi desenvolvido para exibir a rota de coleta no mapa utilizando a integração com o Google Maps. Esse componente foi criado para atender às necessidades específicas do projeto, que não seriam totalmente supridas pelos widgets padrão disponíveis no Flutter. A funcionalidade principal do widget é desenhar as linhas poligonais que representam a rota no mapa, além de posicionar marcadores nas localizações das lixeiras ao longo do trajeto. A customização foi essencial para adaptar o Google Maps às necessidades do projeto, garantindo que a visualização da rota seja clara, funcional e dinâmica, baseada nos dados fornecidos pelo sistema.

Integração com a API

O aplicativo se comunica com o backend por meio de APIs REST, enviando dados de geolocalização em tempo real e recebendo as rotas otimizadas.

4.3 Simulação com Node-RED

Neste projeto, o Node-RED foi utilizado para simular o comportamento de lixeiras inteligentes, devido à complexidade que seria acrescenta ao projeto para aquisição e configuração de sensores físicos. A ferramenta possibilitou a criação de fluxos que replicam o envio de dados em tempo real, como volume de resíduos e peso, além de eventos como a coleta dos resíduos. Esses fluxos foram configurados para reproduzir o comportamento de 10 lixeiras distintas, integradas ao sistema por meio do protocolo MQTT. Abaixo, detalhamos os fluxos e seus respectivos nós, conforme representado na figura [5].

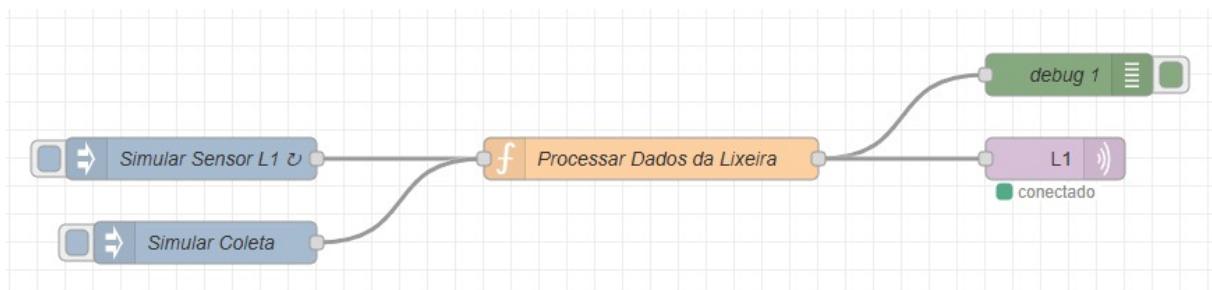


Figura 5: Fluxo do Node-RED

4.3.1 Nós de Entrada

- Simular Sensor L1
 - Nó *Inject* configurado para disparar automaticamente a cada 10 segundos.
 - Simula a geração contínua de dados do sensor, incluindo o volume preenchido e o peso acumulado da lixeira.
- Simular Coleta
 - Nó *Inject* acionado manualmente para simular a coleta da lixeira, enviando uma mensagem com msg.coleta = true.

4.3.2 Processamento dos Dados

O nó "Processar Dados da Lixeira" realiza o tratamento das informações recebidas da seguinte forma:

- Define propriedades fixas, como o ID da lixeira e coordenadas de latitude e longitude.
- Incrementa o volume e o peso de forma aleatória, simulando o acúmulo de resíduos.
- Zera os valores de volume e peso caso a coleta seja simulada.
- Monta uma mensagem com os dados processados (volume, peso, timestamp) para publicação.

A seguir, o código X ilustra a função aplicada no nó 'Processar Dados da Lixeira', responsável por tratar as informações conforme descrito anteriormente.

Função 1 – Processa dados da lixeira

```

1  let lixeira = {
2      id: "L1",
3      latitude: -16.603826,
4      longitude: -49.257907
5  };
6
7  let volumeAtual = context.get("volumeAtual") || 0;
8  let pesoAtual = context.get("pesoAtual") || 0;
9
10 volumeAtual += (Math.random() * 5) + 1;
11
12 pesoAtual += 1000 * (Math.random() * 0.01);
13
14 if (volumeAtual > 200 || pesoAtual > 200) {
15     msg.coleta = true;
16 }
17
18 if (msg.coleta) {
19     volumeAtual = 0;
20     pesoAtual = 0;
21 }
22
23 context.set("volumeAtual", volumeAtual);
24 context.set("pesoAtual", pesoAtual);
25
26 lixeira.volumeAtual = parseFloat(volumeAtual.toFixed(2));
27 lixeira.pesoAtual = parseFloat(pesoAtual.toFixed(2));
28 lixeira.timestamp = Date.now();
29
30 msg.payload = lixeira;
31 return msg;

```

4.3.3 Nós de Saída

- *Debug*
 - Exibe os dados processados no painel do Node-RED para monitoramento.
- L1 (MQTT Out)
 - Publica os dados simulados em um tópico MQTT, permitindo a integração com o backend do sistema.

5 Funcionamento do Sistema

5.1 Aplicativo Móvel

A seguir, descreveremos o fluxo de funcionamento do sistema e as principais funcionalidades implementadas no aplicativo móvel. Serão apresentados os casos de uso que ilustram as

interações dos agentes com o sistema, acompanhados de exemplos visuais que demonstram as telas e os fluxos correspondentes.

Os vídeos que demonstram o funcionamento do aplicativo podem ser acessados nos seguintes links:

- [Clique aqui para o vídeo demonstrativo do aplicativo em um smartphone](#)
- [Clique aqui para o vídeo demonstrativo do aplicativo em um tablet](#)
- [Clique aqui para o vídeo demonstrativo da continuidade do aplicativo em diferentes dispositivos](#)

1. Login

O fluxo do sistema se inicia com a tela de login (figura 6), onde o agente insere suas credenciais para acessar o aplicativo. Embora seja uma funcionalidade básica, ela é essencial para garantir a identificação do usuário, autenticação e possíveis auditorias.

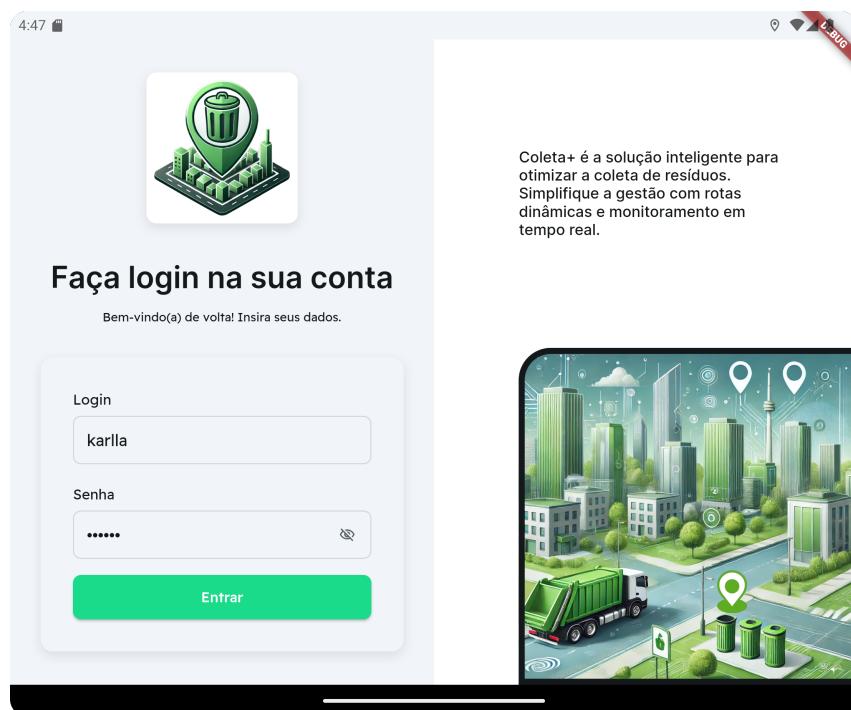


Figura 6: Login do aplicativo

2. Buscar rotas

Ao entrar no aplicativo, o agente tem a opção de buscar a melhor rota disponível para iniciar sua coleta (figura 7, a)). Caso seja a primeira vez utilizando o aplicativo, o sistema solicita permissão para acessar a localização do dispositivo (figura 7, b)), garantindo a geração de rotas otimizadas com base na posição atual do agente.

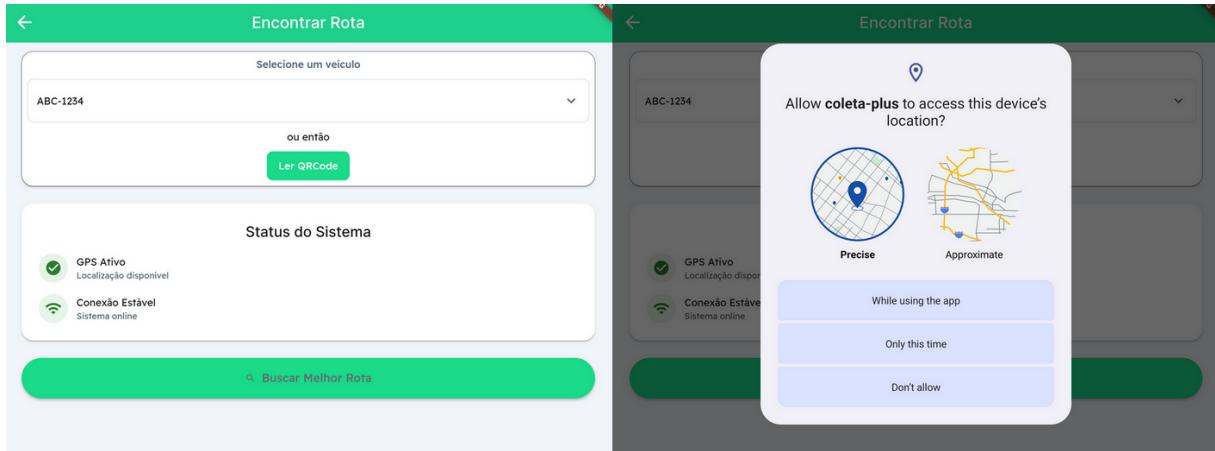


Figura 7: a) Buscar rotas. b) Permissão de acesso à localização do aplicativo

3. Iniciar coleta

Após selecionar a rota, o sistema exibe uma revisão detalhada, destacando informações como o número de lixeiras na rota, a distância total prevista e outros dados relevantes para o agente. Essa etapa permite que o usuário revise os pontos de coleta antes de iniciar o turno, garantindo clareza e previsibilidade para as próximas etapas.

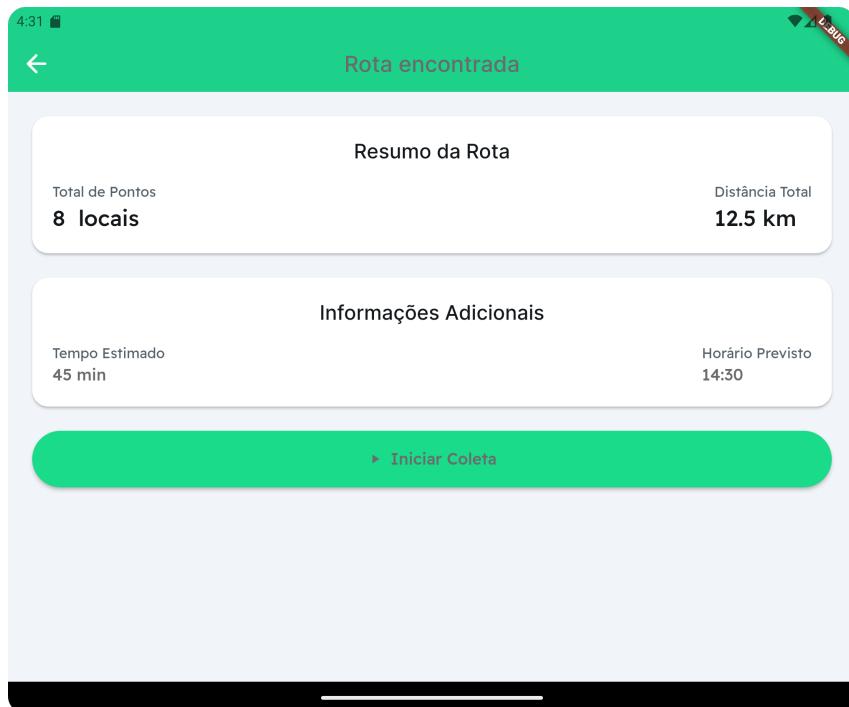


Figura 8: Resumo da rota de coleta

4. Visualização de rotas otimizadas

O agente pode acessar a rota otimizada gerada pelo sistema, onde as lixeiras são organizadas por prioridade com base no nível de preenchimento e proximidade, conforme

figura 9. A visualização inclui um mapa dinâmico, além de informações adicionais como a distância até a próxima lixeira.

A interface do aplicativo é responsiva, adaptando-se ao dispositivo em uso. Em smartphones na orientação vertical, o aplicativo exibe apenas o mapa para facilitar a navegação. Já em tablets ou smartphones na orientação horizontal, ele apresenta informações complementares, como uma lista lateral de lixeiras na rota, indicando o status de cada uma (coletada ou pendente).

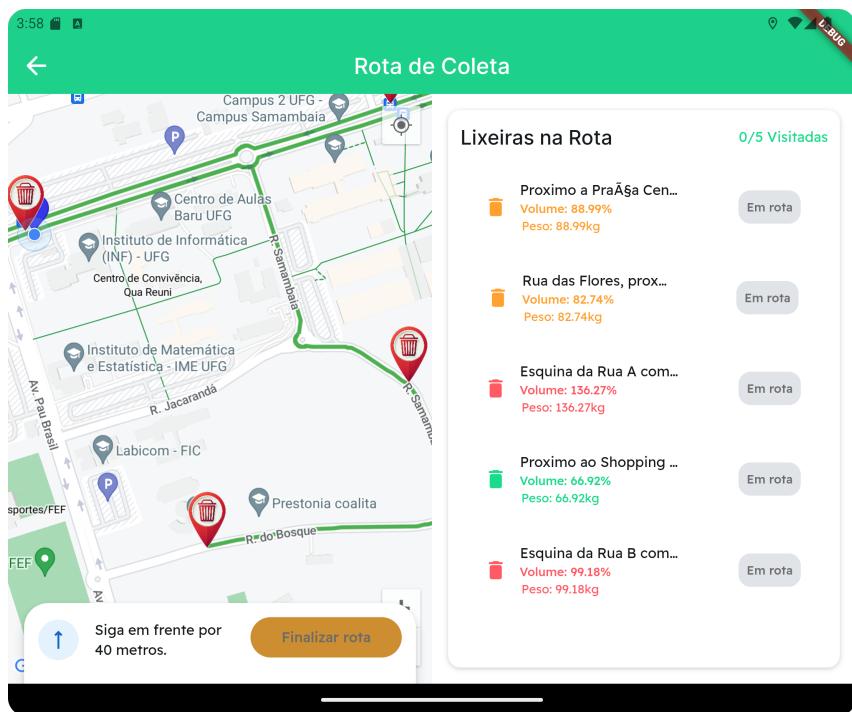


Figura 9: Visualização do trajeto

5. Notificação de mudança na rota

Durante a coleta, o sistema monitora continuamente o estado das lixeiras e a localização do agente e, ao detectar que uma nova lixeira atingiu capacidade crítica, o sistema recalcula a rota automaticamente.

O aplicativo, então, notifica o agente sobre a atualização e anova rota é exibida em tempo real, conforme pode ser visualizado na figura 10.

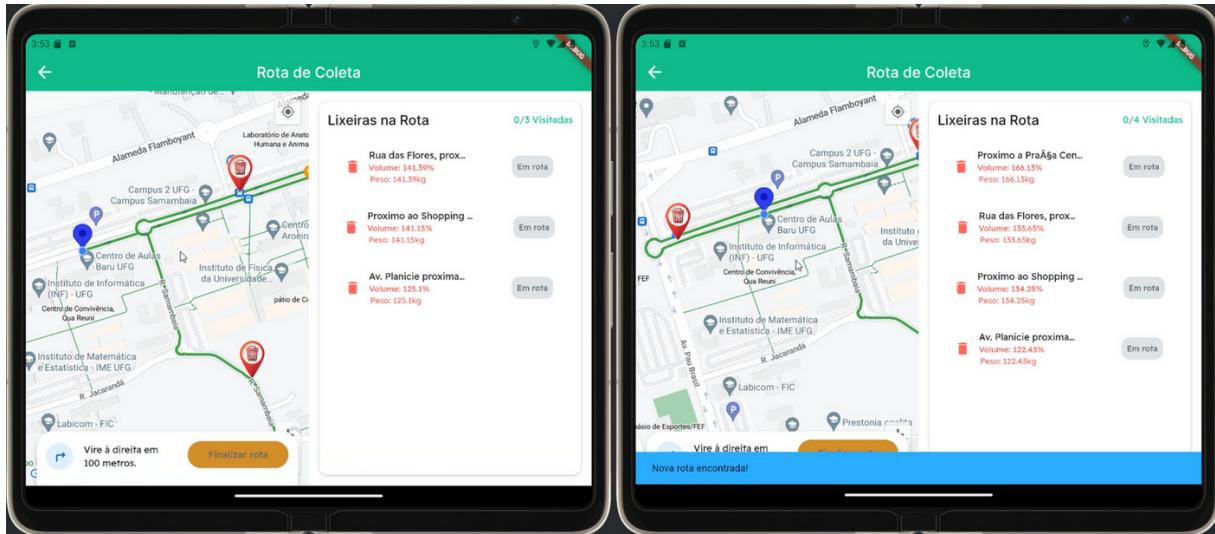


Figura 10: Notificação no aplicativo

6. Registro de coleta de lixeiras

O sistema identifica automaticamente quando uma lixeira foi coletada, utilizando a geolocalização do agente e a redução significativa no nível de resíduos como critérios. Assim que o evento é detectado, o status da lixeira no aplicativo é atualizado para "visitada" (Imagem 11). Após a confirmação da coleta, o status é alterado para "coletada", garantindo um acompanhamento preciso e detalhado do progresso da rota.

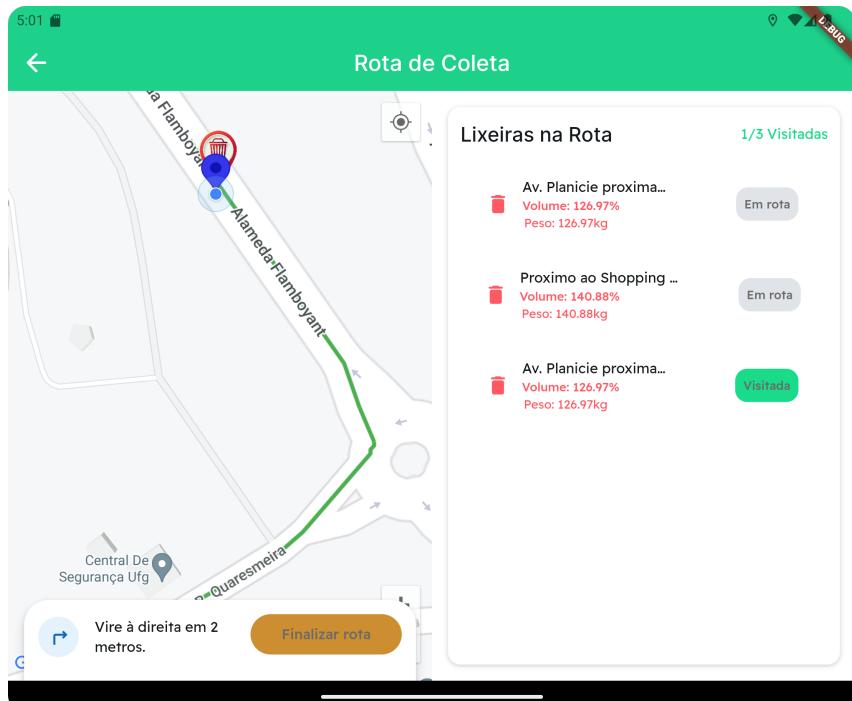


Figura 11: Lixeira visitada

5.2 Sistema de Gestão e Relatórios

O módulo de relatórios foi desenvolvido para fornecer aos supervisores uma visão detalhada sobre a eficiência do sistema e o comportamento das coletas realizadas. Ele apresenta dados como distância percorrida, tempo gasto, emissão de carbono e consumo de combustível dos veículos, permitindo análises gerenciais e tomadas de decisão baseadas em evidências. A seguir, estão exemplos visuais que ilustram os relatórios gerados pelo sistema, destacando sua funcionalidade e usabilidade.

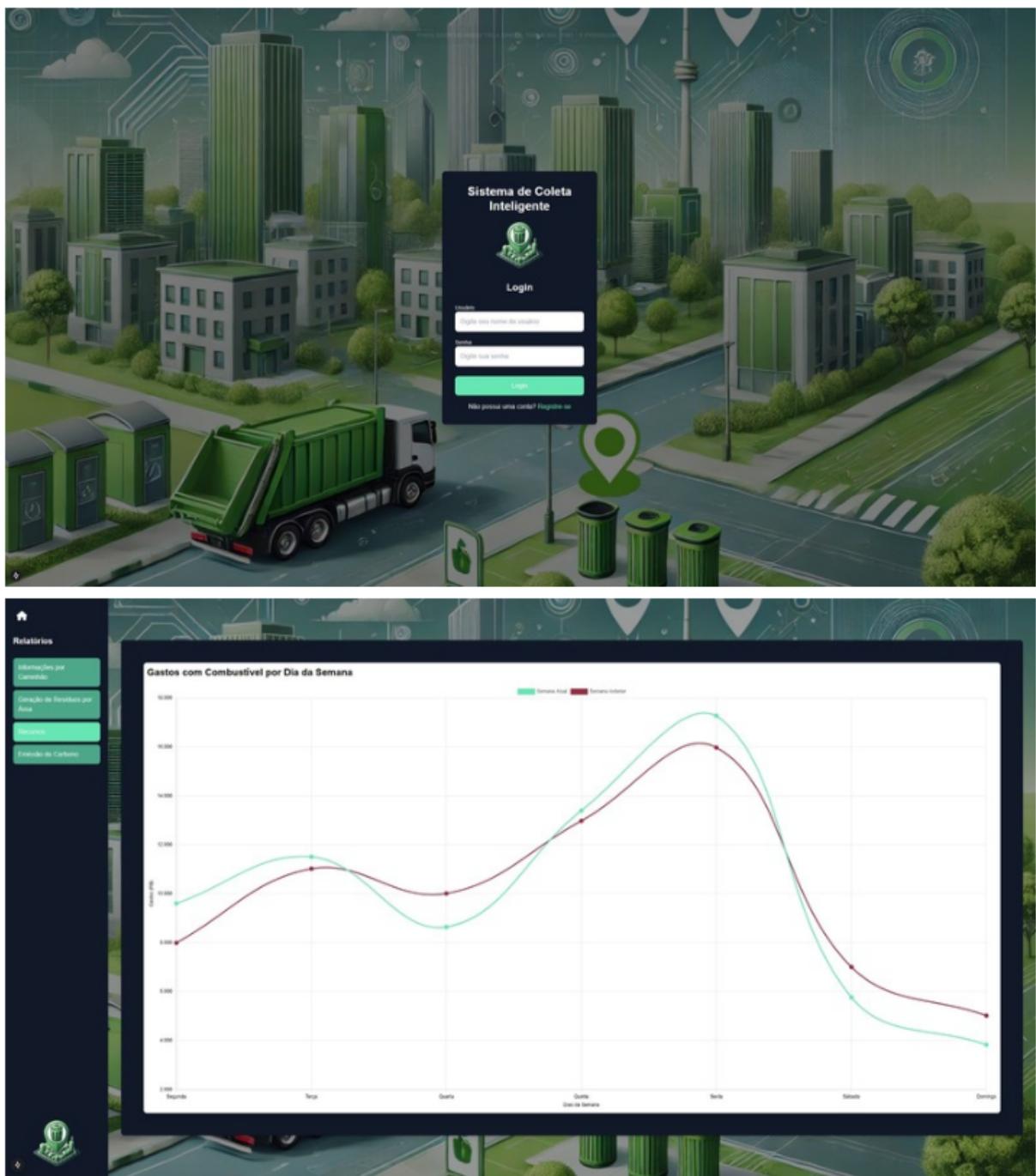


Figura 12: Portal web - Login e consumo dos veículos

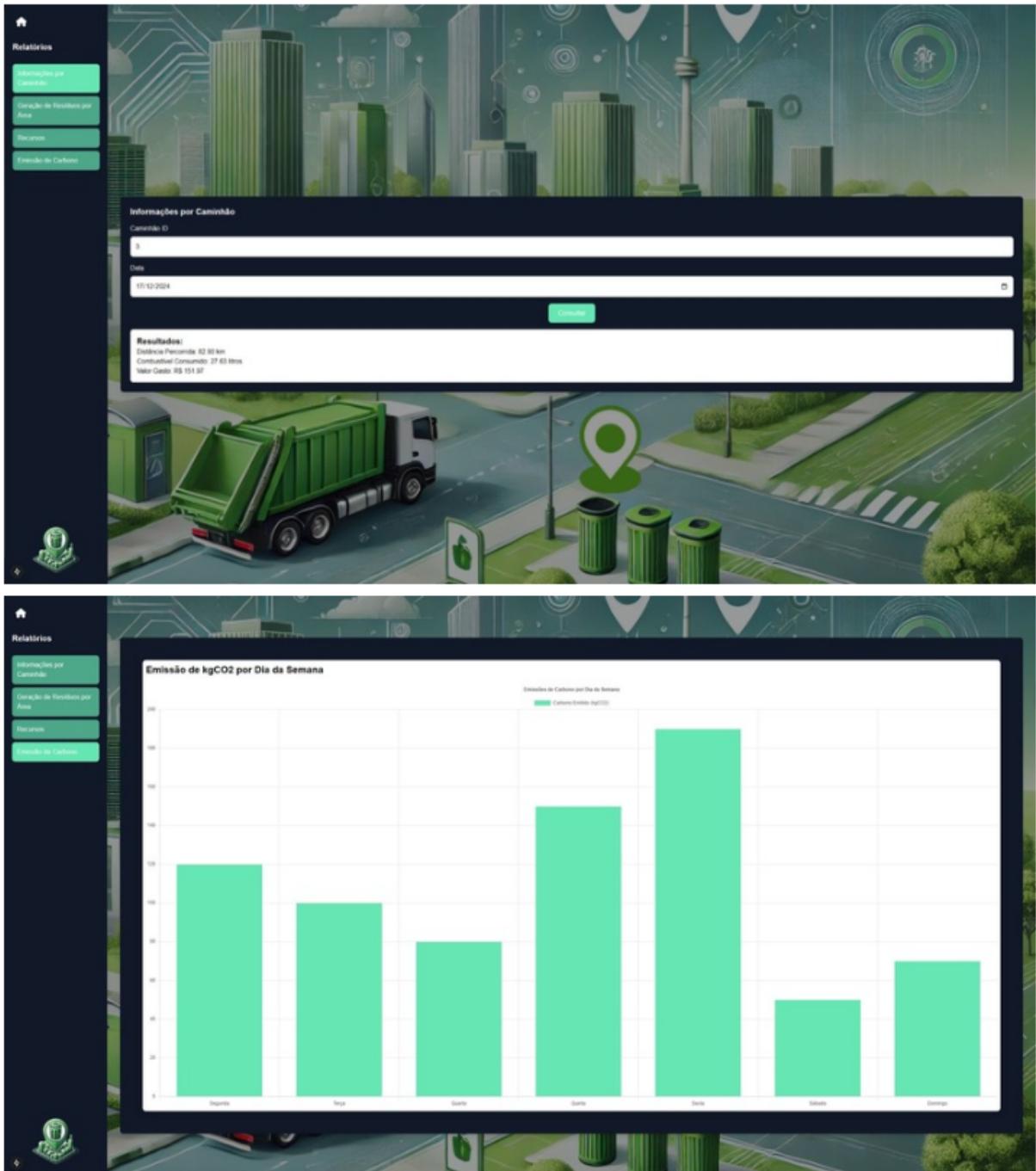


Figura 13: Portal web - Relatório de emissão de C02

6 Conclusão e Trabalhos Futuros

O sistema desenvolvido demonstrou a viabilidade de otimizar a coleta de resíduos em ambientes urbanos inteligentes, integrando tecnologias como IoT, processamento em tempo real e computação ubíqua. Por meio de sensores simulados, o sistema foi capaz de monitorar o estado das lixeiras, gerar rotas dinâmicas e fornecer dados relevantes para agentes e supervisores. O uso de ferramentas como Node-RED, MQTT, e APIs REST assegurou uma arquitetura escalá-

vel e adaptável, enquanto as interfaces responsivas garantiram uma experiência consistente para os usuários.

Para os trabalhos futuros, o sistema pode ser aprimorado com a integração de sensores físicos reais, substituindo as simulações do Node-RED para validar seu desempenho em condições reais de operação. Além disso, a implementação da funcionalidade de continuidade de sessão permitirá que os usuários alternem entre dispositivos sem interrupções, melhorando a experiência de uso e alinhando-se aos princípios da computação ubíqua. Outra melhoria inclui a adoção de algoritmos avançados de otimização de rotas, que levem em conta fatores adicionais como trânsito ou restrições geográficas, para aumentar ainda mais a eficiência das coletas. Por fim, a evolução da arquitetura para incluir edge computing possibilitará o pré-processamento de dados diretamente nos dispositivos IoT ou em gateways locais, reduzindo a latência e melhorando o desempenho em tempo real do sistema.

Referências

- [1] Objetivo 11: Tornar as cidades e os assentamentos humanos inclusivos, seguros, resilientes e sustentáveis. <https://brasil.un.org/pt-br/sdgs/11>. Acessado em: 18 de dezembro de 2024.
- [2] Objetivo 13: Ação contra a mudança global do clima. <https://brasil.un.org/pt-br/sdgs/13>. Acessado em: 18 de dezembro de 2024.
- [3] Liang Gu, Danhuai Zeng, Shenghui Song, and Guowei Xu. Static offloading strategy for mobile cloud computing based on cost model. In *2016 IEEE International Conference on Cloud Computing and Big Data*. IEEE, 2016.
- [4] Xunyun Liu, Amir Vahid Dastjerdi, and Rajkumar Buyya. Stream processing in iot: Foundations, state-of-the-art, and future directions. In Rajkumar Buyya and Amir Vahid Dastjerdi, editors, *Internet of Things: Principles and Paradigms*. Elsevier, 2016.

A História de Usuários e Cenários

HU1 - Geração automática de rotas otimizadas

Como agente de coleta,

Quero que o sistema gere automaticamente uma rota com base na minha localização e no volume das lixeiras,

Para garantir uma coleta eficiente e sem desperdício de recursos.

Cenário 1.1: Geração de rota no início do turno

Dado que iniciei meu turno através do aplicativo,

E o sistema detectou lixeiras com necessidade de coleta,

Quando o sistema processar as informações,

Então uma rota otimizada é enviada automaticamente para o meu dispositivo, priorizando as lixeiras mais cheias.

Cenário 1.2: Inclusão de lixeira cheia durante a rota

Dado que estou executando uma rota ativa,

E uma nova lixeira atinge a 70% de sua capacidade,

Quando o sistema receber essa alteração,

Então ele recalculará a rota e a enviará para o meu dispositivo, com a lixeira incluída no novo trajeto.

Cenário 1.3 (Negativo): Falha no envio da rota

Dado que iniciei meu turno,

E o sistema não conseguiu processar as informações devido à falta de conexão,

Quando eu tentar atualizar a rota,

Então o sistema exibirá uma mensagem de erro e sugerirá uma rota alternativa baseada nos últimos dados disponíveis.

HU2 - Notificação de mudanças na rota em tempo real

Como agente de coleta,

Quero ser notificado automaticamente sobre mudanças na rota,

Para garantir que eu atenda às novas demandas de coleta de lixo.

Cenário 2.1: Lixeira cheia identificada durante a coleta

Dado que estou seguindo a rota ativa,
E uma lixeira próxima atinge 70% de sua capacidade,
Quando o sistema detectar a situação e recalcular a rota,
Então eu recebo uma notificação com a rota atualizada.

Cenário 2.2 (Negativo): Notificação atrasada ou não recebida

Dado que estou executando uma rota ativa,
E uma lixeira atinge 80% de sua capacidade,
Quando o sistema recalcula a rota mas não consegue enviar a notificação,
Então eu continuo com a rota anterior e o sistema registra a falha para supervisão.

HU3 - Selecionar o caminhão que realizará a coleta

Como agente de coleta,
Quero selecionar o caminhão que será utilizado para realizar a coleta,
Para que o sistema registre e associe o veículo às rotas designadas.

Cenário 3.1: Seleção bem-sucedida do caminhão

Dado que iniciei meu turno através do aplicativo,
E o sistema exibe a lista de caminhões disponíveis,
Quando eu selecionei o caminhão que será utilizado,
Então o sistema registra a associação do caminhão com as rotas designadas para o turno.

Cenário 3.2 (Negativo): Falha na seleção do caminhão

Dado que iniciei meu turno pelo aplicativo,
E o sistema não consegue acessar os dados de disponibilidade dos caminhões devido a um erro de conexão,
Quando eu tentei selecionar o caminhão,
Então o sistema exibe uma mensagem informando a falha e sugere tentar novamente ou contatar o suporte.

HU4 - Iniciar coleta no aplicativo

Como agente de coleta,
Quero iniciar meu turno pelo aplicativo,

Para que o sistema registre minha localização e gere a rota de coleta automaticamente.

Cenário 4.1: Início de turno pelo aplicativo

Dado que estou com o aplicativo aberto,
E o sistema já identificou que existem lixeiras a serem coletadas,
Quando eu inicio a coleta no aplicativo,
Então o sistema registra minha localização inicial e gera uma rota otimizada para coleta.

Cenário 4.2 (Negativo): Falha no registro de localização

Dado que estou tentando iniciar meu turno pelo aplicativo,
E o sistema não consegue acessar minha localização,
Quando eu tento registrar a coleta,
Então ele exibe uma mensagem de erro e solicita que eu ative o GPS para prosseguir.

HU5 - Gerar relatórios de eficiência da coleta

Como supervisor,
Quero ter acesso a relatórios de coleta de lixo,
Para avaliar o desempenho ecológico e operacional das rotas sugeridas.

Cenário 5.1: Geração de relatório de coleta

Dado que estou acessando o portal do sistema,
Quando eu solicitar o relatório no sistema,
Então ele apresentará dados como distância percorrida, tempo gasto e a emissão de carbono durante a coleta.

Cenário 5.2 (Negativo): Falha na geração de relatório

Dado que estou solicitando um relatório,
E o sistema não consegue processar os dados devido a um erro no servidor,
Quando eu tento gerar o relatório,
Então o sistema exibe uma mensagem informando a falha e sugere uma nova tentativa em poucos minutos.

HU6 - Visualizar tendências de geração de resíduos

Como supervisor,

Quero visualizar gráficos de tendência sobre a geração de resíduos por região,

Para identificar áreas com maior frequência de coleta e sazonalidades.

Cenário 5.1: Gráfico de tendência por região

Dado que o sistema possui dados históricos das coletas,

E cada lixeira está associada a uma localização específica,

Quando eu acesso o dashboard de tendências,

Então eu visualizo um gráfico com regiões que geram mais resíduos e o sistema sugere ajustes na frequência de coleta.

Cenário 5.2 (Negativo): Dados insuficientes para análise

Dado que o sistema possui poucos dados históricos de coleta,

Quando eu acesso o dashboard de tendências,

Então o sistema exibe um aviso informando que os dados disponíveis podem não refletir tendências confiáveis.

HU7 - Visualizar mapa com o estado atual das lixeiras

Como gestor,

Quero visualizar um mapa da região com o estado atual das lixeiras,

Para monitorar a capacidade de cada lixeira e planejar coletas futuras.

Cenário 7.1: Visualização de mapa atualizado

Dado que estou acessando o portal do sistema,

E o sistema possui dados atualizados dos sensores,

Quando eu seleciono a opção de visualizar o mapa,

Então ele exibe um mapa da região com o estado atual de cada lixeira, indicando aquelas próximas à capacidade máxima.

Cenário 7.2 (Negativo): Dados insuficientes para análise

Dado que estou tentando acessar o mapa,

E o sistema não consegue carregar os dados devido a uma falha de conexão,

Quando eu solicito a visualização,

Então o sistema exibe uma mensagem de erro informando o problema e sugere uma nova tentativa mais tarde.

HU8 - Visualizar histórico de coleta de uma lixeira

Como gestor,

Quero visualizar o histórico de coleta de uma lixeira específica,

Para analisar padrões de uso e identificar necessidades de ajustes na frequência de coleta.

Cenário 8.1: Consulta de histórico de coleta

Dado que estou acessando o portal do sistema,

E selecionei uma lixeira específica,

Quando eu solicito o histórico de coleta,

Então o sistema exibe os registros de coleta daquela lixeira, incluindo datas, horários e volumes coletados.

Cenário 8.2 (Negativo): Dados insuficientes para análise

Dado que estou tentando consultar o histórico de uma lixeira,

E o sistema não possui dados registrados para aquela lixeira,

Quando eu solicito o histórico,

Então o sistema exibe uma mensagem informando que não há dados disponíveis para análise.

B Links úteis

O controle de versões foi realizado por meio da plataforma GitHub, com a organização dos repositórios dividida da seguinte forma:

- **Repositório backend:** [sicoin-backend](#)
- **Repositório do frontend:** [sicoin-frontend](#)
- **Repositório para aplicativo mobile:** [scu-coletaplus](#)
- **Repositório para versionamento do documento de arquitetura e diagramas:** [PadraoArqui-SCU](#)
- **Documento de arquitetura:** [PadraoArqui-SCU](#)