UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS DE INFORMAÇÃO

FRANCISCO SACCO FLORES ALMEIDA TEIXEIRA

RENOVAR: UM MVP PARA MONITORAR A QUALIDADE DO AR

Florianópolis - Santa Catarina
2018

FRANCISCO SACCO FLORES ALMEIDA TEIXEIRA

RENOVAR: UM MVP PARA MONITORAR A QUALIDADE DO AR

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação, da UNIVERSIDADE FEDERAL DE SANTA CATARINA, como requisito parcial para a Obtenção do grau de Bacharel em Sistemas de informação.

Florianópolis - Santa Catarina
2018

FRANCISCO SACCO FLORES ALMEIDA TEIXEIRA

RENOVAR: UM MVP PARA MONITORAR A QUALIDADE DO AR

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação, da UNIVERSIDADE FEDERAL DE SANTA CATARINA, como requisito parcial para a Obtenção do grau de Bacharel em Sistemas de informação.

Florianópolis - Santa Catarina, 05 de dezembro de 2018

BANCA EXAMINADORA

Prof. Dr. José Eduardo De Lucca
Universidade Federal de Santa Catarina

Profº Dr. Leonardo Hoinaski
Universidade Federal de Santa Catarina

Profº Dr. João Candido Lima Dovicchi
Universidade Federal de Santa Catarina

Profº Dr. Mario Antonio Ribeiro Dantas
Universidade Federal de Santa Catarina

Dedico este trabalho a meus pais, Cícero e Rejane, aos meus irmãos, Rodrigo e Bárbara e ao meu amor, Bruna, por sempre estarem presentes e serem os presentes.

AGRADECIMENTOS

Ao longo desses últimos dois anos eu tive inúmeras pessoas que trocaram experiências, conhecimento, tempo, espaço e diversos momentos que possibilitaram a construção desse trabalho de conclusão de curso. Não atribuo o pronome possessivo “meu” para este, pois foi construído a partir de muitas partes, cuja soma é muito maior que o todo.

Este trabalho representa a conclusão de uma etapa que é consideravelmente importante, uma vez que a conclusão de um curso de graduação não atinge nem 20% dos brasileiros. É uma honra poder ter tido acesso a um ensino público excelente e de qualidade, portanto meu agradecimento inicial é a Universidade Federal de Santa Catarina e o Estado por ter me possibilitado essa oportunidade.

Eu também sou muito grato a sociedade como um todo, pois em um sistema tributário como o Brasil possui, ela auxiliou a investir no conhecimento que adquiri nos últimos 5 anos da minha vida, além de investir em toda a infraestrutura humana e física que permite o funcionamento da UFSC.

“Renovar: um MVP para monitorar a qualidade do ar” é nosso, construído por muita gente que precisa ser lembrada. É possível que esqueça alguns nomes, por isso desde já meu mais sincero muito obrigado a todos: família, amigos e conhecidos que fizeram parte desta etapa. O conhecimento é uma das poucas coisas que compartilhada, aumenta.

Agora preciso enfatizar o poder de estar presente, assim como na dedicatória também agradeço a Cícero, meu pai, professor de muitos e muitas, educador de outros tantos. Nunca deixou de ir em uma reunião de escola, jamais esqueceu um aniversário, sempre lembrou que a vitória está na mente e que nunca deve se desistir do que começou. Agradeço pelas lições de equilíbrio que foram muito mais além da bicicleta. Plim.

Rejane, minha mãe, também professora e educadora de centenas de crianças e adolescentes. Desde muito cedo aprendeu a lutar pela vida e me ensinou o significado dessa luta. Agradeço por me ensinar as direções e confiar na direção que escolhi para a minha própria vida.

Rodrigo, meu irmão, um amante do drama e da comédia, sempre capaz de me mostrar como trazer esses artifícios para dar mais graça, sentido e esperança a vida, a rotina e ao futuro. Bárbara, simplesmente, me inspiro na tua força e agradeço por me mostrar que é importante continuar sorrindo não importando a situação.

Bruna, me mostraste o significado do amor e de amar. Agradeço pela tua compreensão, carinho e pelas injeções de inspirações que me fazem constantemente querer ser melhor. Obrigado por aceitar meus silêncios e por encontrar as minhas palavras em dias turbulentos. Obrigado por ser essa mulher que luta por um mundo mais igual, acessível e justo.

Também gostaria de agradecer à BRy Tecnologia, pela oportunidade e confiança desde o início do meu estágio em 2015, até minha efetivação dois mais tarde, possibilitando meu amadurecimento pessoal e profissional. Agradeço Ramon, Leandro, Eduardo e Janira pelo convívio e troca diária de experiências. Também gostaria de agradecer especialmente ao Fabyan, por todos os ensinamentos, boas práticas, e pela sua conduta moral e ética na qual me inspiro.

Os meus amigos de curso que compuseram a gestão do Centro Acadêmico de Sistemas de Informação entre 2016 e 2017: Bruno, Jan, Júlia, Jan, Leo, Leon, Luiza e Richard. Agradeço o convite para participar e ajudar na construção de uma entidade muito importante para os estudantes e para a universidade. Sou muito

grato pela amizade que fizemos durante a nossa gestão e pela confiança que criamos uns com os outros.

Aos meus colegas ingressos na universidade junto comigo no segundo semestre de 2013, meu profundo agradecimento. Contudo dou ênfase aqueles cuja amizade tornou-se mais forte no decorrer da graduação: Júlia, Marcelo, Péricles e Vitor. Agradeço por viverem a universidade comigo fora das salas de aula, por construírem um pensamento mais crítico e progressista, por compartilhar histórias e experiências para fortalecer o nosso lado humano, agradeço o apoio em vários momentos e obrigado pela importante base para que eu chegasse até aqui. Vocês tem um espaço enorme nessa trajetória, são dignos das posições que ocupam, mais uma vez muito obrigado.

As amizades que surgiram nos corredores, no centro acadêmico e aos arredores da UFSC, quero agradecer imensamente a Gustavo e Guilherme. Vocês estiveram comigo em vários momentos, fomos cúmplices de histórias para uma vida toda e foram ombros que muitas vezes me ajudaram em momentos difíceis.

Ao Núcleo de Humanização Artes e Saúde, Humanizarte e Terapeutas da Alegria, do qual participei no início da graduação e que proporcionou experiências únicas e amizades que guardarei com muito carinho, agradeço em especial: Pamela, Renê, Larissa, Barbara, João, Gabriela e Thiago

Ao Laboratório de Controle e Qualidade de Ar da UFSC, agradeço toda a estrutura e oportunidade de construir um projeto ao lado de uma equipe muito competente. Gostaria de enfatizar meu agradecimento a Igor, Victoria, Fernando e ao professor Leonardo por acompanharem de perto todo o desenvolvimento do projeto.

Ao professores da banca Dovichi e Dantas que mesmo estando longe, estiveram presentes e puderam contribuir com a avaliação desse trabalho. Também gostaria de agradecer ao meu orientador professor José Eduardo De Lucca pela amizade, confiança e acima de tudo compreensão no decorrer de todo desenvolvimento deste trabalho.

A Laura, Lucila e Machado uma família para além da terra natal. Sou realmente muito grato por todo o carinho e pela receptividade. Uma amizade que transcende gerações e capaz de sempre oferecer um abraço fraterno a qualquer momento.

À Dona Oliveira, jamais esquecerei de ti, uma segunda mãe, que esteve comigo desde pequeno e acompanhou todo meu crescimento. Agradeço por sempre incentivar o estudo e se preocupar comigo e com meu irmão.

Ao Cardoso, Gabriel, Nicolas e Vinícius agradeço pela amizade sincera, por estarmos juntos independente da distância, por conseguir perpetuar esse sentimento para além do tempo e espaço. Agradeço pelas conversas, pelas viagens e pela lembrança de um tempo cuja vida era mais simples.

A Thomas, amigo de longa data. Obrigado por me ensinar que dias ruins sempre existirão, mas se já soubermos disso é possível aceitar e saber conviver com eles.

Por fim, reitero meu agradecimento a todos.

"Navegar é preciso, viver não é preciso."

(Fernando Pessoa)

RESUMO

A Internet das Coisas, do inglês Internet of Things (IoT), vem crescendo cada vez mais com a popularização da tecnologia no dia a dia das pessoas. IoT é um conceito que reúne em um único guarda-chuva diversas tecnologias e sistemas para oferecer uma maior integração das "coisas" físicas com o mundo digital. Para tal, é utilizado sensores, acionadores e comunicação de dados (normalmente sem fio) para estabelecer essa conexão/integração, possibilitando monitorar dados em tempo real, processá-los, informá-los e alertar usuários em situações específicas. Diversas áreas são beneficiadas com a introdução de dispositivos, sensores e gerentes de ambientes e informações. Um dos segmentos da IoT, é cidades inteligentes, ambiente com alta capacidade inovativa o qual utiliza de dados sobre a cidade para monitorá-la. Dentre as muitas áreas utilizadas para considerar uma cidade inteligente, o meio ambiente é uma das mais importantes, pois impacta diretamente na qualidade de vida da população. Um dos fatores que mais implicam na qualidade de vida é a qualidade do ar, que segundo a OMS é considerado o maior risco ambiental e causador de mais de 7 milhões de mortes somente em 2016. Dessa forma, com o uso de sensores a diversidade de dados que é possível coletar torna-se muito maior, sobretudo dados relacionados a qualidade do ar. A partir da preocupação com o meio ambiente, observando os altos índices de poluição que influenciam diretamente na qualidade de vida da população, surge a premissa da plataforma Renovar: o desenvolvimento de uma rede cidadã para controlar indicadores de qualidade do ar, através da participação dos cidadãos em prol de uma cidade, cada vez mais, inteligente.

Palavras-chave: IoT, cidades inteligente, meio ambiente, ciência cidadã.

ABSTRACT

The Internet of Things, has been growing more and more with the popularization of technology in people's daily lives. IoT is a concept that brings together in a single umbrella diverse technologies and systems to offer a greater integration of physical "things" with the digital world. To do this, sensors, triggers and data communication (usually wireless) are used to establish this connection / integration, allowing real-time monitoring, processing, informing and alerting users in specific situations. Several areas benefit from the introduction of sensor devices and environment and information managers. One of the segments of IoT is smar cities, an environment with high innovative capacity, which uses data about the city to monitor it. Among the many areas used to consider an intelligent city, the environment is one of the most important, as it directly impacts the quality of life of the population. One of the factors that most implicate quality of life is air quality, which according to the WHO is considered the greatest environmental risk and causes more than 7 million deaths in 2016. Thus, with the use of sensors, the diversity of data that it is possible to collect becomes much larger, mainly data related to air quality. Observing the high pollution indexes that directly influence the quality of life of the population, there is the premise of the Renovar platform: the development of a citizen network to control air quality indicators through of citizens' participation in favor of an increasingly smart city.

Key words: IoT, smart cities, environment, citizen science.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CONAMA	Conselho Nacional do Meio Ambiente
CSS	Cascading Style Sheets
DTO	Data Transfer Object
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IoT	Internet of Things
JDK	Java Development Kit
JPQL	Java Persistence Query Language)
JSON	JavaScript Object Notation
LCQAR	Laboratório de Controle e Qualidade do Ar
MIME	Multipurpose Internet Mail Extensions
MP	Material Particulado
MVC	Model, View and Control
MVP	Minimum Value Product
NPM	Node Package Manager
OMS	Organização Mundial da Saúde
ONU	Organização das Nações Unidas
REST	Representational State Transfer
SASS	Syntactically Awesome Stylesheets
TCC	Trabalho de Conclusão de Curso
UFSC	Universidade Federal de Santa Catarina
V	Volts
WHO	World Health Organization

SUMÁRIO

1 **INTRODUÇÃO**

Nos últimos anos, as tecnologias móveis se tornaram cada vez mais acessíveis, tornando o mundo um lugar mais conectado. Não somente *smartphones*, computadores, *tablets* tem acesso à internet, mas uma nova tendência tecnológica chamada Internet das Coisas, (IoT) vem surgindo, possibilitando uma conexão muito mais ampla e diversa, contemplando um "guarda-chuva" de dispositivos.

A IoT já está trazendo um grande impacto na sociedade, segundo reportagem BNDES (2017) a IoT pode gerar ganhos potenciais em 70 bilhões de dólares na economia nacional até 2022. Esse impacto é grande, em função da grande abrangência que a IoT permite em diversos setores econômicos e sociais. A partir de um estudo publicado recentemente pela IoT Analytics[1], observa-se uma crescente preocupação no desenvolvimento de aplicações relacionadas a IoT e subdividindo-se em vários segmentos como: Indústria, Saúde, Casas, Cidades Inteligentes, Agricultura, Mobilidade, Meio Ambiente.

Cidades inteligentes, mais precisamente, é um dos segmentos que merece destaque, pois abrange diversos outros, incluindo a IoT. As cidades inteligentes tem o objetivo de integrar dados dessas áreas a fim de otimizar processos, criar políticas públicas, gerando mais qualidade de vida a população. Dentre os segmentos que compõem as cidades inteligentes, o meio ambiente precisa ser enfatizado visto que é um dos poucos aspectos que está presente em diversas pesquisas como: European Smart Cities, Ranking Connected Smart Cities, Iese Cities in Motion e Br-SCMM.

A partir disso essas três áreas: IoT, Cidades Inteligentes e Meio Ambiente, são capazes de gerar um grande insumo de conhecimento quando conectadas, proporcionando um impacto social significativo e de empoderamento populacional. A IoT proveniente da evolução tecnológica é elo entre muitos setores, justamente por conseguir produzir dados, através de sensores, para a tomada de decisão, alertar e informar entidades interessadas sobre determinado aspecto; neste caso, cidades inteligentes e meio ambiente.

[1]O artigo completo está disponível em https://iot-analytics.com/global-overview-1600-enterprise-iot-projects/

1.1 PROBLEMA

Segundo Silva, Ferreira e Santos (2015) a problemática ambiental atualmente é uma preocupação para a comunidade científica. O crescimento da urbanização está atrelado a intensificação das atividades socioeconômicas, segundo a ONU metade da população mundial, em torno de 3,6 bilhões de pessoas, vivem em cidades e a perspectiva é que em 2050 esse número suba para 6 bilhões de pessoas.

Ainda, o aumento da concentração populacional também eleva o desenvolvimento de setores de produção e transporte, por meio da combustão de gases responsáveis por alterar o balanço energético da atmosfera das cidades (SILVA; FERREIRA; SANTOS, 2015). Essa combustão gera a emissão de poluentes, principalmente monóxido de carbono (CO), hidrocarbonetos (HC) e óxidos de nitrogênio (NOx), bem como a resuspensão de material particulado do solo. Tais elementos, quando em alta concentração, são conhecidos por causarem sérios danos à saúde dos seres vivos, causando mortes precoces e internações hospitalares. Por esse motivo, a poluição atmosférica é considerada o maior risco global à saúde ambiental (UNEP, 2014).

Segundo IBGE (2013), Florianópolis é a terceira capital com a maior frota de carros per capita do país. Se por um lado esse dado representa um indicador de bom desenvolvimento econômico, em contra partida apresenta-se como um vetor de deterioração da qualidade do ar da cidade, visto que a maioria da frota é movida à queima de combustíveis fósseis. Conforme a World Health Organization (2015; 2016), outro aspecto negativo, é a incidência de doenças causadas por poluição do ar, ainda, a poluição atmosférica foi causa de 7 milhões de mortes no ano de 2016, além de apresentar uma série de resultados[2] importantes para a saúde populacional.

Para mitigar o problema de poluição atmosférica, o Brasil conta com a resolução da CONAMA nº03/1990. A qual, sobretudo, apresenta quais são os níveis adequados dos poluentes e os limites de concentração que estes podem estar expostos no ambiente a fim de não danarem a saúde humana. Apesar de a resolução estar em vigor há mais de 25 anos, Florianópolis não possui monitoramento da qualidade do ar.

[2]Estatísticas na íntegra https://www.who.int/airpollution/infographics/en/

Observa-se então uma possível despreocupação quanto a qualidade do ar da cidade e observou-se o quanto este se apresenta nos últimos anos uma problemática crítica para a saúde global, fazendo-se necessário, dessa forma, que haja informações atualizadas constantemente sobre a qualidade do ar. Para tal, sugere-se nesse estudo o desenvolvimento de *Renovar*: um MVP para a coleta e monitoramento de dados ambientais na IoT, de baixo custo, colaborativo e que visa auxiliar a longo prazo no desenvolvimento de uma cidade inteligente.

1.2 OBJETIVOS

1.2.1 **Objetivo Geral**

O objetivo desse projeto é o desenvolvimento de um protótipo de coleta e monitoramento da qualidade do ar, disponibilizando esses em domínio público, por meio de uma solução IoT e da pesquisa e implementação de uma solução de hardware e software livre, de baixo custo e colaborativo. Esse projeto foi nomeado Renovar, inicialmente, desenvolvido e testado como um MVP, aplicado em Florianópolis para posterior escala.

1.2.2 **Objetivos Específicos**

- Desenvolver um módulo para coletar e armazenar dados de sensores que medem a qualidade do ar;
- Desenvolver um módulo web para visualizar os relatórios a partir dos dados coletados;
- Propor uma documentação de um modelo escalável e colaborativo a partir dos módulos previamente desenvolvidos.

1.3 JUSTIFICATIVA

A poluição atmosférica é um problema que se apresenta com elevados custos sociais e econômicos. Silva, Ferreira e Santos (2015) apontam que as áreas urbanas não planejadas sofrem demasiadamente pelo processo rápido de industrialização, assim como o grande fluxo de transporte urbano. Florianópolis possui uma frota per capita significativa e uma população urbana concentrada, intensificada na temporada turística. Estudos realizados no Laboratório de Controle

da Qualidade do Ar – LCQAr da Universidade Federal de Santa Catarina apontam que as concentrações de alguns poluentes atmosféricos podem violar os padrões de qualidade do ar em alguns pontos da cidade, principalmente no entorno da Avenida Beira Mar Norte. Além de ser a principal via de ligação da cidade, também é um local que favorece a prática de atividade físicas.

Por mais que algumas medidas já estão sendo tomadas para controlar os índices de poluição no mundo, tais como: promover políticas de desenvolvimento sustentável que apoiem a qualidade do ar saudável, divulgar causas de poluição atmosférica interna e externa relacionando-as com doenças, apresentar os elevados custos das mortes provenientes da poluição atmosférica (World Health Organization, 2015). Em Santa Catarina, estado do sul brasileiro, não foi encontrado nenhuma iniciativa tomada pelo poder público estadual e municipal para responder estas demandas. A capital catarinense, Florianópolis, não possui monitoramento da qualidade do ar, contrariando inclusive a resolução CONAMA n°03/1990.

Em paralelo a isso, a tecnologia digital evolui exponencialmente, podendo-se observar avanços nas mais diversas áreas. A IoT é uma delas, que para Borgia (2014) é um novo paradigma na qual se intersecta aspectos e tecnologias com variadas abordagens. São fundidas diversas tecnologias as quais são distribuídas em alguns escopos, tais quais: protocolos de Internet, sensores, tecnologia de comunicação, computação ubíqua, computação pervasiva e dispositivos embarcados que ainda demandam armazenagem e processamento de grandes quantidades de dados. Essas se unem com o objetivo de formar um sistema no qual a realidade e a virtualidade estão em um sincronismo simbiótico.

A partir dessas explanações, o projeto proposto buscou implementar uma rede de coleta e disponibilização de dados sobre poluição, com foco inicial em qualidade do ar. Para a realização de testes foi instalado, uma estação de monitoramento no LCQAr, a fim de coletar dados meteorológicos e de poluição e enviá-los para servidores da Internet. Esses dados foram publicados sobre mapas em plataforma aberta e também, contou com uma metodologia simplificada para interpretação desses dados e de como eles podem influenciar na saúde.

1.4 METODOLOGIA

O presente TCC apresenta-se como uma pesquisa de caráter descritivo, que correlaciona aspectos que envolvem fatos ou fenômenos, neste caso, questões dentro da área ambiental vinculados ao contexto da IoT assim como cidades inteligentes. Para a primeira etapa faz-se necessário:

- Levantamento bibliográfico documental para compreender o contexto da área de estudo e suas dificuldades;
- Apresentação do estado da arte da IoT aplicado ao monitoramento de dados de qualidade ambientais.

A segunda parte do trabalho envolve um estudo prático. O desenvolvimento de um MVP, em tradução direta Mínimo Produto Viável, que conforme Ries (2011) é o produto mais rápido e econômico de ser desenvolvido, a fim de expô-lo ao mercado no mais curto ciclo de projeto possível. A terceira e última parte do trabalho, sugere um artefato documental para a continuidade e replicabilidade e escalabilidade do MVP desenvolvido.

1.5 ESCOPO

O TCC não contempla o desenvolvimento de um novo padrão para medir a qualidade do ar, contudo baseia-se na Resolução CONAMA n. 3 (1990) que elucida os limites de poluentes do ar relacionados à poluição. O dispositivo utilizado para a coleta de dados foi desenvolvido em parceria com o LCQAr, Laboratório de Controle e Qualidade do Ar do Departamento de Engenharia Sanitária e Ambiental da Universidade Federal de Santa Catarina. Não sendo ênfase o trabalho com sensores, mas sim a comunicação entre esses.

O trabalho não objetivou a análise dos dados, uma vez que foi implementado em escala reduzida e em local estratégico, entretanto enfatizou o relacionamento de áreas de estudo: meio ambiente e sistemas de informação. Sendo essa aplicação um MVP, o qual não visou fins comerciais, mas objetiva uma contribuição inicial para o controle da qualidade do ar e meio ambiente, inicialmente, em torno da UFSC.

2 **FUNDAMENTAÇÃO TEÓRICA**

Nesta etapa são apresentados e conceituados os temas relacionados ao presente trabalho de conclusão de curso, por intermédio de uma pesquisa bibliográfica.

> A pesquisa bibliográfica é o primeiro passo de um trabalho científico. Por meio dela é possível coletar e verificar a parte teórica sobre os temas e assuntos que serão de interesse no andamento do trabalho científico. (DMITRUK, 2001)

2.1 INTERNET DAS COISAS

Segundo Zambarda (2014), a ideia de conectar objetos é discutida desde 1991, quando a conexão TCP/IP e a Internet, como é conhecida hoje, começou a se popularizar. Em 1999, Kevin Ashton do MIT propôs o termo “Internet das Coisas" e dez anos depois escreveu o artigo “A Coisa da Internet das Coisas” para o RFID Journal. Zambarda (2014) ainda diz que a “Internet das Coisas” se refere a uma revolução tecnológica que tem como objetivo conectar os equipamentos usados no dia a dia à rede mundial de computadores.

Conforme Atzori, Morabito e Iera (2010), quando se intersseciona os conceitos de internet e coisas, eles assumem um significado que traz um disruptivo nível de inovação para as tecnologias de comunicação e informação. O autor também diz que semanticamente, Internet das Coisas é uma rede de objetos interconectados que possuem endereçamento único, baseado em um padrão de protocolos de comunicação. Esses objetos estão conectados a internet e podem “sentir”, controlar, analisar e decidir automaticamente, de forma distribuída e colaborativa com outros objetos. Ainda, os objetos relativos a IoT também podem rastrear, identificar a localização, monitorar e gerenciar uns aos outros (SALEEMA et al., 2016).

2.2 SENSORES

Para Priberam, a palavra *sensor* conceitua-se como um “dispositivo que permite adquirir, ler ou transmitir uma informação". De acordo com Loureiro et al. (2003), os dispositivos têm a capacidade de sensoriar, processar e comunicar. Em uma rede de sensores, cada um, ou cada grupo de sensores pode exercer tarefas

diferentes: sensoriamento do ambiente, processamento da informação e tarefas associadas com tráfego de dados.

2.3 ARQUITETURA REST

Segundo Codecademy, REST é um estilo de arquitetura para prover padrões entre sistemas de computadores e a web, tornando mais fácil a comunicação entre as partes. Codecademy explica que *REST-compliant systems* também podem ser chamados como RESTful systems, ou seja sistemas caracterizados como aplicações sem estado, *stateless,* e que separam as preocupações do cliente e do servidor.

Em uma arquitetura do estilo REST, as implementações do cliente e do servidor podem ser feitas independentes uma das outras. Isso significa que o código do lado do servidor pode ser atualizado sem interferir no cliente e que o código do cliente pode ser alterado sem afetar o servidor (Codecademy). É importante lembrar que cada lado tem de saber qual formato de mensagem deve enviar e receber, dessa forma tanto o cliente como o servidor podem ficar modulares, separados e restritos às suas responsabilidades. O cliente, por exemplo, responsável pela interface do usuário e o servidor no armazenamento de dados, pode permitir que cada componente melhore independentemente.

2.3.1 **Cliente**

O cliente pode ser qualquer tecnologia que realiza uma requisição para o *back-end*. Normalmente os clientes são os navegadores (Google Chrome, Firefox, Opera) que fazem requisições a partir de códigos HTML e JavaScript para acessar websites (Codecademy). Entretanto, existem muitos outros tipos de clientes: uma aplicação mobile, uma aplicação em outro servidor, e também um dispositivo inteligente conectado a internet.

2.3.2 **Servidor**

Um servidor é um computador que recebe requisições, porém há máquinas que são desenvolvidas especificamente para esta finalidade.

2.3.3 **Stateless**

Para Codecademy sistemas que seguem o paradigma REST são *stateless,* ou seja, o servidor não precisa salvar nenhuma informação sobre o estado do cliente e vice-versa. Dessa forma, ambos podem entender qualquer mensagem sem a necessidade de haver um contexto pré-estabelecido.

2.3.4 **Comunicação Cliente e Servidor**

Em uma arquitetura REST, clientes enviam requisições para adicionar, recuperar ou modificar recursos, e os servidores enviam respostas para essas requisições. De acordo com Codecademy, as requisições são formadas conforme a lista a seguir:

- Verbo HTTP, que define o tipo de informação que será realizada;
- *Header* (cabeçalho), que permite o cliente enviar alguma informação sobre a requisição;
- Caminho do recurso, que também é chamado de *endpoint* ou *path;*
- Uma *message body (*corpo da mensagem*),* parte opcional utilizada para enviar dados.

2.3.4.1 Verbos HTTP

Há quatro principais verbos HTTP, que podem ser escolhidos para montar uma requisição:

- GET, recebe um recurso ou coleção destes;
- POST, cria um novo recurso;
- PUT, atualiza um recurso, normalmente, a partir de um identificador;
- DELETE, remove um recurso a partir de um identificador.

2.3.4.2 *Header*

De acordo com Codecademy no *header* de uma requisição há o campo *Accept,* em que o cliente envia o tipo de conteúdo, MIME Types, que será aceito pelo servidor. Isto garante que o servidor só envie dados que serão entendidos e processados pelo cliente.

2.3.4.3 *Endpoint*

Uma requisição contém um *Endpoint* ou *Path,* para acessar um recurso no qual a operação deve ser executada. Nas APIs RESTful os *endpoints* devem ser projetados para auxiliar o cliente a saber o que está acontecendo. Recomenda-se que a primeira parte do *path* deve estar no plural, desta forma os caminhos derivados deste ficam de simples e fácil entendimento.

Os caminhos devem conter as informações necessárias para localizar um recurso com o grau de especificidade necessário. Ao se referir a uma lista ou coleção de recursos em requisições GET, não é necessário adicionar um identificador a uma solicitação, assim como nas requisições POST, pois o servidor gerará um identificar para o novo objeto.

2.4 CIÊNCIA CIDADÃ

De acordo com Comandulli et al. (2016), a Ciência Cidadã é entendida como a participação de amadores, voluntários e entusiastas em projetos científicos. Contudo, essa participação limita-se, pois muitas vezes os papéis destinados aos participantes se restringem à observação e coleta de dados. Dessa forma, eles não participam da definição dos problemas e nem da análise científica. Por sua vez, conforme Jenkins (1999), Ciência Cidadã é aquela que está relacionada com termos de reflexão, ou seja, as preocupações, interesses e atividades dos cidadãos das tarefas do seu cotidiano.

2.5 CIDADES INTELIGENTES

Nos últimos anos o termo Cidade Inteligente, do inglês Smart City, ganhou destaque. Para Komninos (2013), uma cidade inteligente é capaz de conectar o digital e o real, já para Bouskela et al. (2016) a tecnologia conecta cidadãos, empresas à cidade e entre si. Ambos autores dizem que uma cidade inteligente está associada diretamente com um espaço delimitado, realizando uma gestão em prol das tecnologias de informação e comunicação, para diminuir as ilhas de informação e realizar uma distribuição inteligente de recursos.

Uma cidade inteligente possui grande capacidade inovativa, formada por uma considerável produção intelectual das suas populações, aplicando a gestão do conhecimento e da comunicação para melhorar a estrutura de serviço da cidade, consequentemente, melhorando a qualidade de vida dos cidadãos (KOMNINOS,

2013). Bouskela et al. (2016), afirma que além da melhora da gestão de comunicação, há também a melhora da eficiência de operações, serviços urbanos e de sua competitividade.

Uma cidade inteligente visa promover a qualidade de vida da população, através do desenvolvimento econômico, por oferta de estrutura básica em saúde, saneamento, água, educação e preservação do meio ambiente (JÚNIOR; GALIOTTO, 2013). Para que seja possível o aumento da qualidade de vida, é preciso incorporar ao conceito de *smart city* aspectos relativos à governança, à infraestrutura e ao capital humano e social. Só é possível tornar uma cidade efetivamente inteligente quando esses elementos são agregados entre si, para promover desenvolvimento sustentável e integrado (BOUSKELA et al., 2016).

As tecnologias são grandes aliadas das cidades inteligentes, entre elas estão as redes de conectividade de banda larga e alta velocidade fixas e móveis, a coleta de dados, análise de dados, aplicações móveis, mídias sociais, portais web, entre outras ferramentas. Esse conjunto de ferramentas cria um contingente de cidadãos conectados e os inclui em uma gestão participativa (BOUSKELA et al., 2016), podendo, dessa forma fazer uma associação com ciência cidadã, justamente por incluir os participantes na coleta, análise e tomada de decisão (COMANDULLI et al., 2016).

O Bouskela et al. (2016), ainda diz que uma ferramenta muito importante para acompanhar dados em tempo real é a utilização de sensores distribuídos em vários pontos na cidade os quais fornecem dados sobre o fluxo de cidadãos, nível de ruído e outras formas de poluição ambiental, assim como tráfego e condições climáticas. Dessa forma, é possível que as autoridades responsáveis otimizem as operações da cidade, incluindo melhor gestão ambiental, otimização da mobilidade urbana, sustentabilidade econômica e social.

2.5.1 **Indicadores de Cidades Inteligentes**

Junckes e Teixeira (2016), elucidam alguns trabalhos relacionados a quais indicadores são necessários para analisar a performance de uma cidade e considerá-la inteligente. A seguir é listado esses trabalhos e quais as suas principais abordagens.

2.5.1.1 European Smart Cities

No continente europeu, realizou-se um trabalho em conjunto com o Centro de Ciência Regional da Universidade de Tecnologia de Viena na Áustria, o Departamento de Geografia da Universidade de Liubliana, Eslovênia, e o Instituto de Pesquisa para Habitação, Urbanismo e Estudos de Mobilidade da Universidade de Tecnologia Delft, na Holanda. O trabalho foi o desenvolvimento de uma metodologia para analisar a performance das cidades, sob os aspectos referentes à: economia inteligentes, pessoas inteligentes, governança inteligente, mobilidade inteligente, meio ambiente inteligente, vida inteligente (Vienna University Technology, 2015).

2.5.1.2 Ranking Connect Smart Cities

O Ranking Connected Smart Cities foi desenvolvido por duas organizações: a Urban System e a Sator. Para construí-lo as duas empresas mapearam as principais publicações nacionais e internacionais, em 2014, sobre cidades inteligentes, cidades sustentáveis e cidades conectadas. A parceria resultou em um estudo englobando 11 segmentos: Mobilidade, Urbanismo, Meio Ambiente, Energia, Tecnologia e Inovação, Economia, Educação, Saúde, Segurança, Empreendedorismo e Governança (CONNECT SMART CITIES, 2017).

2.5.1.3 Iese Cities in Motion

Estudo realizado pela Universidade de Navarra na Espanha que leva em consideração mais de 50 indicadores entre 10 diferentes dimensões: Governança, Gestão Pública, Planejamento Urbano, Tecnologia, Meio Ambiente, Relações Internacionais, Coesão Social, Mobilidade e Transporte, Economia, Pessoas (IESE, 2014).

2.5.1.4 Br-SCMM

Br-SCMM é o Modelo Brasileiro de Maturidade para Cidades Inteligentes, que vem sendo desenvolvido através da disponibilidade de dados em domínios públicos sobre cidades brasileiras, comparadas com análises realizadas em outras localidades do mundo. Os domínios que são levados em consideração nesse

modelo são: Educação, Saúde, Água, Energia, Governança, Segurança, Meio Ambiente, Urbanização, Tecnologia e Transporte (AFONSO et al., 2013).

2.5.1.5 Indicadores Relacionados

A partir dos projetos anteriormente descritos, foi desenvolvido a Tabela 1 para apresentar quais dos indicadores são referentes a cada estudo e, também quais indicadores se intersectam em mais de um estudo. Como é possível visualizar na Tabela 1, observa-se que apenas “Meio ambiente” e “Mobilidade e transportes” encontram-se em todos os projetos.

Tabela 1 - Relação entre estudo e indicadores

Indicadores	European Smart Cities	Ranking Connect Smart Cities	Iese Cities in Motion	Br-SCMM
Educação		X		X
Saúde		X		X
Tecnologia		X	X	X
Meio ambiente	X	X	X	X
Mobilidade e transporte	X	X	X	X
Segurança		X		X
Água				X
Economia	X	X	X	
Governança	X		X	X
Energia		X		X
Pessoas	X		X	
Vida	X			
Urbanismo		X	X	X
Relações Internacionais			X	
Empreendedorismo		X		
Coesão social			X	
Gestão Pública			X	

Fonte: O autor (2018)

2.6 MEIO AMBIENTE

Segundo Brasil (2002), meio ambiente pode ser definido como “o conjunto de condições, leis, influencias e interações de ordem física, química, biológica, social, cultural e urbanística, que permite, abriga e rege a vida em todas as suas formas”.

Bouskela et al. (2016) aponta a sua importância, contudo ressalta que a humanidade intensificou as intervenções no meio ambiente, provocando desequilíbrios que apresentam-se cada vez mais constantes e intensos como: inundações, ilhas de calor, concentração de poluentes na atmosfera. Esses desequilíbrios podem influenciar diretamente na qualidade de vida e consequente na saúde da população.

2.7 POLUIÇÃO ATMOSFÉRICA

Para Ferreira e Oliveira (2016), a poluição atmosférica está relacionada a três etapas. A primeira tem o início com a emissão de poluentes por fontes naturais, uma vez que este poluente se encontra na atmosfera, começa a segunda etapa que é o processo de dispersão, no qual é transportado pelos movimentos do ar, precipitações, massas de ar, etc. Na terceira etapa ocorre a interação entre os gases poluentes e a atmosfera o que definirá o grau de qualidade do ar, ou seja, quanto maior a concentração de um ou mais poluentes na atmosfera pior será a qualidade do ar.

Então, de acordo com Azuaga (2000), uma fonte emissora contínua dependerá das características da atmosfera em dissipar ou concentrar mais a poluição. Na atmosfera, a poluição é o resultado do desequilíbrio entre a emissão e a capacidade da atmosfera em dispersar os poluentes (TACO, 2006).

3 **TRABALHOS RELACIONADOS**

Os trabalhos relacionados apresentados neste capítulo trazem aplicações práticas de dispositivos e softwares referentes ao monitoramento de indicadores ambientais, com foco na qualidade do ar. É importante ressaltar que o desenvolvimento desta seção contribuiu consideravelmente para a construção dos capítulos seguintes, pois trouxe inspiração e insumos utilizados na implementação do MVP da plataforma Renovar.

3.1 I-BLADES E ENVIROSENSOR

I-Blades (2018) é uma tecnologia que pode ser acoplada aos *smartphones*, sendo capaz de medir a qualidade do ar, por meio de sensores, no qual o usuário se encontra; ela comunica-se com o aplicativo EnviroSensor para enviar os dados e, este apresentar as informações ao usuário. Essa é uma solução pioneira, pois foi a primeira vez que houve a integração entre um *smartphone* e um sensor de qualidade de ar. Durante um dia, I-Blades pode realizar entre 17 mil a 23 mil medições, mostrando indicadores relacionados tanto à qualidade de ambientes fechados: casas, escritórios, salas de aula, como em ambientes abertos: ruas, parques, praças.

A solução é importante para que o seu usuário conheça o ar que ele respira em todos os ambientes que frequenta, justamente por I-Blades ser portável e flexível. Além disso, alertas são emitidos no aplicativo EnviroSensor quando os indicadores de ar não estão de acordo com o aceitável. I-Blades, possui quatro sensores em um único dispositivo, distribuídos em: Compostos Orgânico Voláteis (COV), temperatura, umidade e pressão. Ressalta-se a importância do sensor COV uma vez que ele mede a concentração de formaldeído, amônia, benzeno, xileno e decano (I-BLADES, 2018).

Figura 1 - I-Blade conectado ao smarphone

Fonte: I-BLADES (2018)

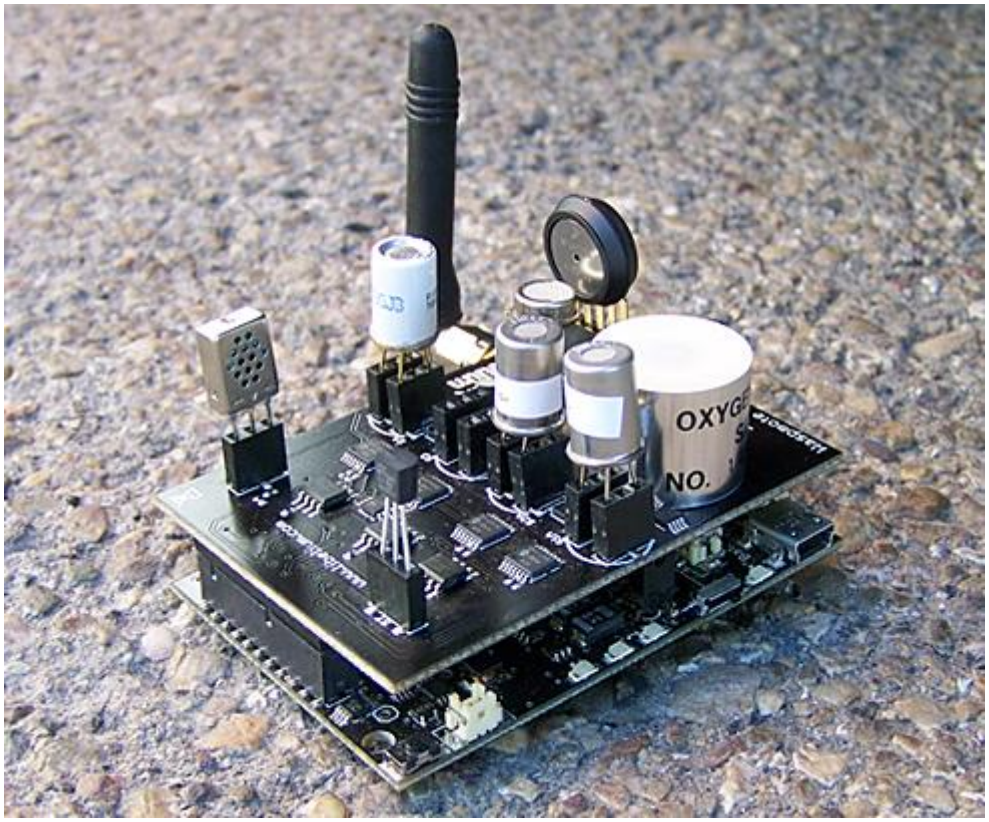
3.2 LIBELIUM - WASPMOTE

Libelium é uma empresa com um foco amplo em IoT. Ela possui tecnologias em vários segmentos: meio ambiente, agricultura inteligente, indústria inteligente, cidades inteligentes, mobilidade e transporte, saúde, educação, entre outras áreas. A empresa oferece um leque variado de sensores relacionados a cada um desses respectivos segmentos.

Na área de meio ambiente a Libelium oferece um sensor chamado Waspmote, desenvolvido a partir da diretiva 96/62 regularizada pela Comissão Europeia, capaz de medir os seguintes parâmetros de qualidade de ar: Dióxido de nitrogênio (NO_2), dióxido de carbono (CO_2), monóxido de carbono (CO), metano (CH_4), sulfato de hidrogênio (H_2S), hidrocarbonetos (etano, propano, butano,

isobutano, tolueno), ozônio (O_3). Além disso, o sensor ainda pode medir radiação ultravioleta, detectar incêndios em florestas e gerenciar a irrigação de parques e jardins. O Waspmote possui placas solares para a recarga de sua baterias e comunicação sem fio via protocolo Zigbee (ASÍN; CALAHORRA, 2010).

Figura 2 - Waspmote



Fonte: Asín e Calahorra (2010)

3.3 DUSTDUINO

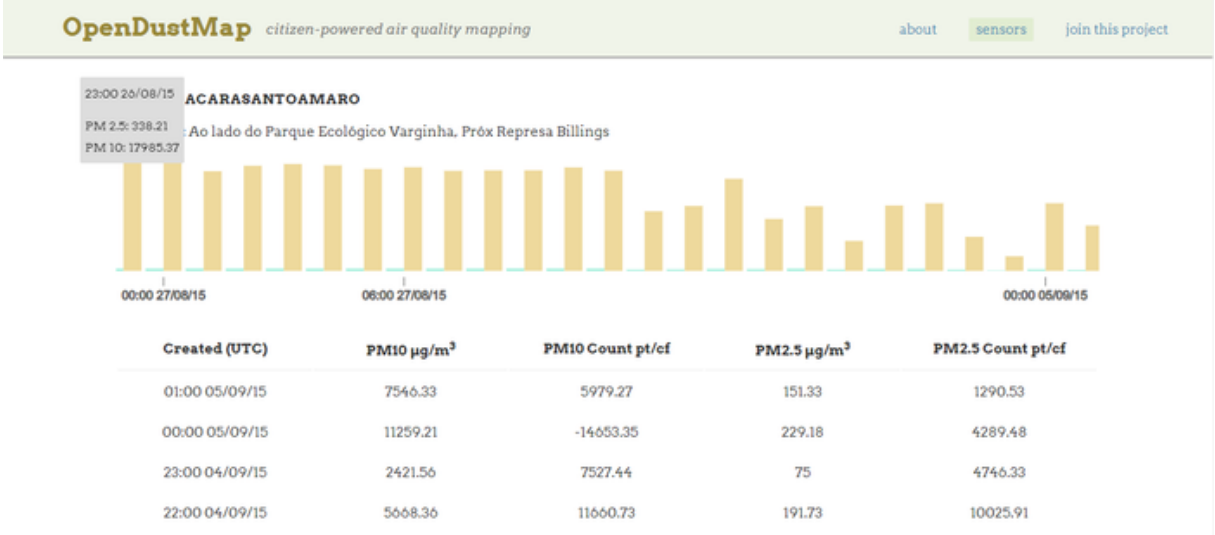
DustDuino é um sensor produzido em parceria por quatro organizações: Earth Journalism Network, Frontline SMS, GroundTruth e Delepment Seed. O sensor é capaz de monitorar a concentração de Material Particulado (MP) do ar, enviando as informações por meio de uma rede 3G ou *Wifi* a cada 40 segundos, para um determinado banco de dados, que armazena e disponibiliza os dados para as entidades responsáveis (Public Lab).

Figura 3 - Sensor DustDuino

Fonte: Santini (2015)

Santini (2015), aborda uma iniciativa no Brasil sobre o uso do DustDuino. Em São Paulo, Garoa Hacker Clube, um grupo de estudos de tecnologia e suas aplicações realizou a instalação do sensor em alguns pontos da cidade com grande circulação de pessoas e automóveis. O objetivo dessa ação foi realizar a comparação dos níveis de poluição em diferentes áreas, para avaliar a evolução e gravidade da concentração de poluentes durante cada época do ano. Os dados coletados foram disponibilizados publicamente para a população ter acesso. A Figura 4 mostra um dos gráficos gerados a partir do dados obtidos.

Figura 4 - Gráfico gerado a partir das coletas do sensor DustDuino



Created (UTC)	PM10 µg/m³	PM10 Count pt/cf	PM2.5 µg/m³	PM2.5 Count pt/cf
01:00 05/09/15	7546.33	5979.27	151.33	1290.53
00:00 05/09/15	11259.21	-14653.35	229.18	4289.48
23:00 04/09/15	2421.56	7527.44	75	4746.33
22:00 04/09/15	5668.36	11660.73	191.73	10025.91

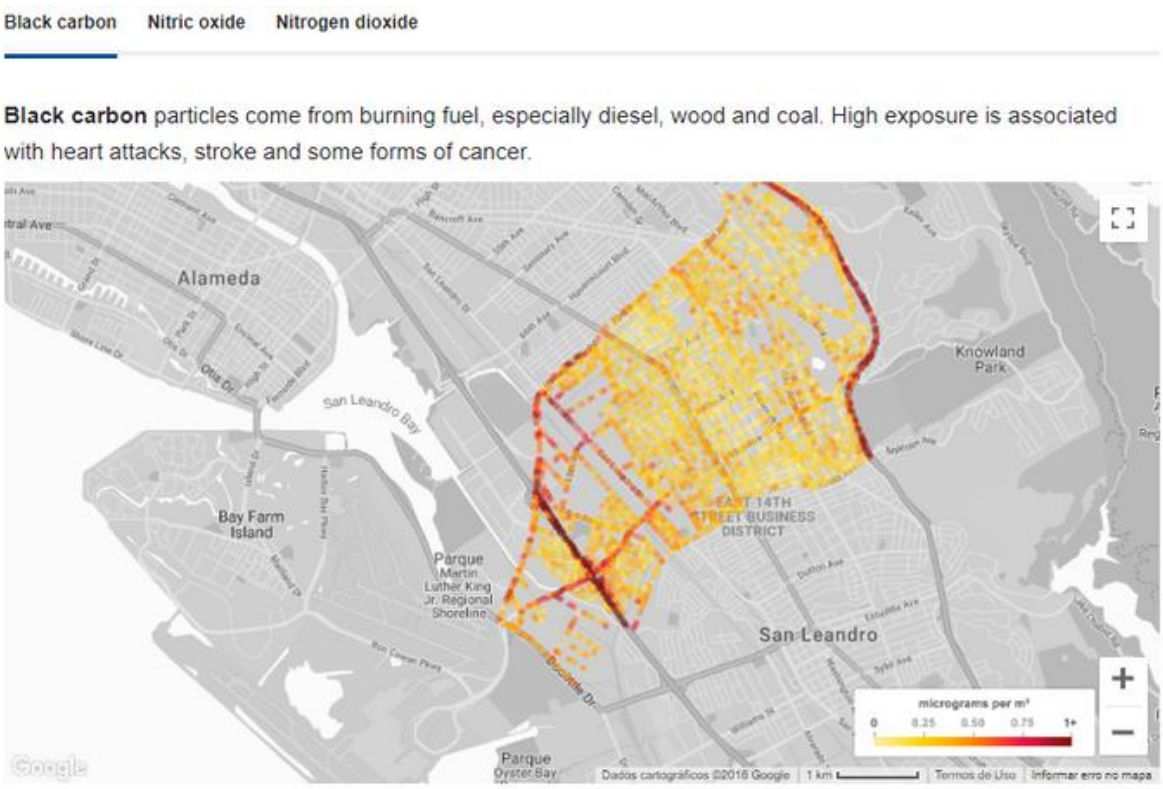
Fonte: OpenDustMap (2015)

3.4 EDF E GOOGLE EARTH OUTREACH

A Google juntamente com EDF (Enviroment Defese Found), desenvolveram um projeto iniciado em 2015 para medir a qualidade do ar por meio de sensores acoplados a um carro da Google, responsável também por fazer o mapeamento do Google Street View. A grande vantagem do projeto é a mobilidade, posto que é possível obter informações sobre a qualidade do ar de vários pontos de uma cidade em um curto espaço de tempo.

A cidade de Oakland, na Califórnia - Estados Unidos, foi a piloto para o projeto, pois em função da grande taxa de doenças respiratórias, câncer, ataques cardíacos, observou-se a necessidade de estudar a causa, refletida nas atividades industriais, automóveis e transporte urbano, consequentemente nos poluentes gerados (Enviroment Defense Found). A Figura 5 mostra a concentração de um dos três tipos de poluentes observados: dióxido de nitrogênio, óxido nítrico e carbono preto.

Figura 5 - Mapeamento de carbono preto na cidade de Oakland



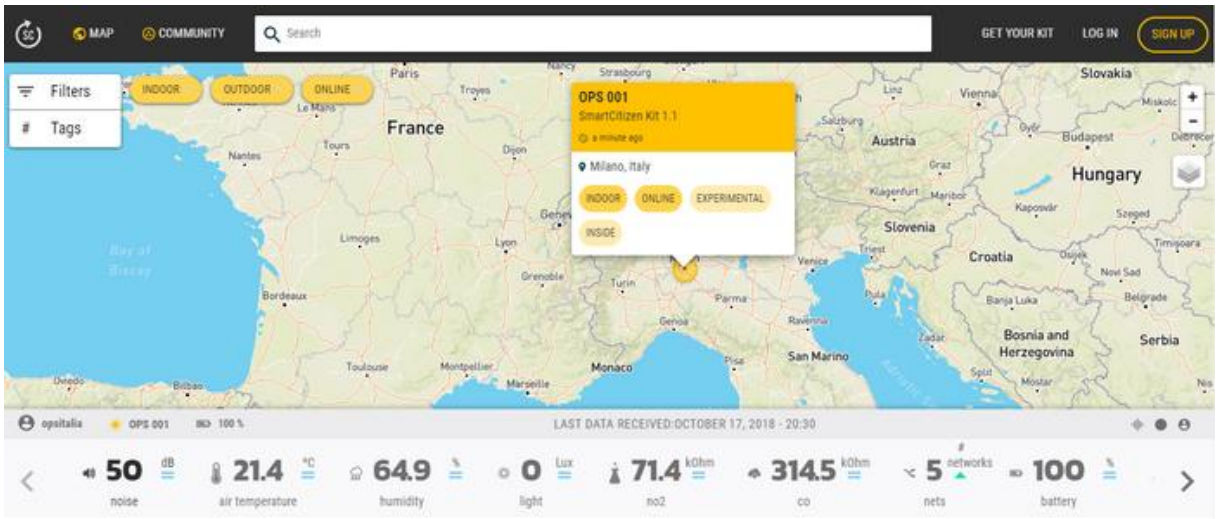
Fonte: Enviroment Defense Found

3.5 SMART CITIZEN

Smart Citizen é uma tecnologia de código aberto para incentivar os cidadãos nos processos participativos de uma cidade inteligente. Apresenta-se como uma plataforma para conectar dados, pessoas e conhecimento, desta forma objetiva atuar como um elo para a construção de indicadores de dados abertos, ferramentas distribuídas e, posteriormente, de uma cidade desenvolvida pelos próprios habitantes.

O projeto é baseado em geolocalização, internet, software e hardwares abertos para coleta e compartilhamento de dados. A partir disso é possível integrar a população com o meio ambiente que ela está inserida, criando uma relação efetivamente melhor entre os recursos, tecnologias, comunidade, serviço, eventos no espaço urbano, público ou particular. A Figura 6 ilustra a interface que o usuário tem ao acessar o mapa disponibilizado pelo projeto, na imagem observa-se os dados coletados em um determinado ponto, neste caso Milano na Itália.

Figura 6 - Interface mostrando indicadores de um sensor



Fonte: Smart Citizen

Os dados coletados foram obtidos por meio de um sensor, responsável por medir composição do ar (monóxido de carbono e dióxido de nitrogênio), temperatura, umidade, intensidade de luz e nível de som. O sensor envia dados por uma rede Wifi e ainda, possui painel solar para realizar a recarga da bateria que consome. A Figura 7 mostra um dos sensores que podem ser utilizados.

Figura 7 - Sensor

Fonte: Smart Citizen

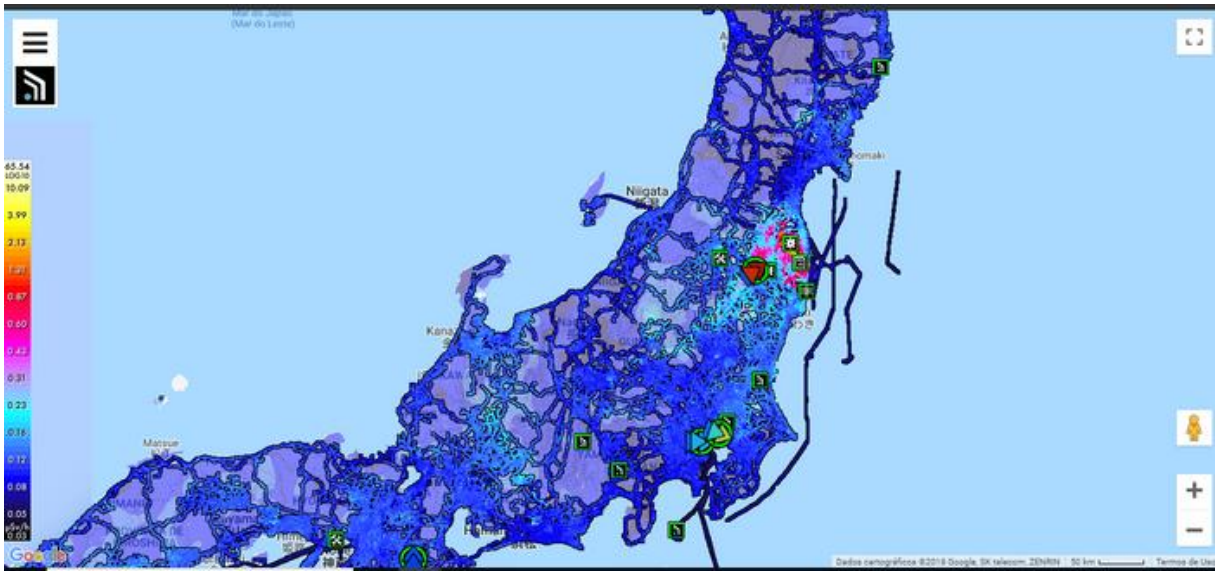
Tendo em vista que o projeto não é uma caixa preta, o usuário pode criar o seu próprio sensor através de Arduino e integrá-lo a plataforma Smart Citizen, assim possibilita-se ter uma maior flexibilidade de desenvolvimento e também escalabilidade (Smart Citizen).

3.6 SAFECAST

Safecast é uma organização internacional voluntária, voltada à ciência cidadã para o meio ambiente. Ela surgiu em março de 2011, após os desastres naturais causados por furacões e tsunamis, ocasionando a crise nuclear em Fukushima no Japão.

A organização tornou-se, rapidamente, uma plataforma para coletar, monitorar e compartilhar informações sobre os níveis de radiação, que naquela época eram demasiadamente prejudiciais. Desde o seu surgimento, a plataforma disponibiliza e organiza abertamente os dados geo localizados. Na Figura 8 é possível observar o mapa do mundo, em destaque o Japão, com o território marcado com cores, quanto mais quente a cor, ou seja: mais próximo do branco, maior o nível de radiação (SAFECAST, 2018).

Figura 8 - Mapa de monitoramento de radiação



Fonte: SAFECAST (2018)

Conforme SAFECAST (2018), além da disponibilização do mapa, a Safecast desenvolveu um dispositivo, Safecast bGeigie Nano, visualizado na Figura 9, que realiza a coleta e envio de dados da radiação para servidores da Internet, por meio de um aplicativo instalado em um *smartphone,* que é sincronizado com o dispositivo. Ele é capaz de monitorar os níveis de radioatividade alfa, beta e gama, enviando dados a cada cinco segundos, ainda, o Safecast bGeigie Nano pode estar

localizado estaticamente, porém também permite esta acoplado a carros, bicicletas, aviões, trens e outros meios de transportes.

Figura 9 - Safecast bGeigie Nano

Fonte: https://blog.safecast.org/bgeigie-nano/

4 **APRESENTAÇÃO DA PROPOSTA**

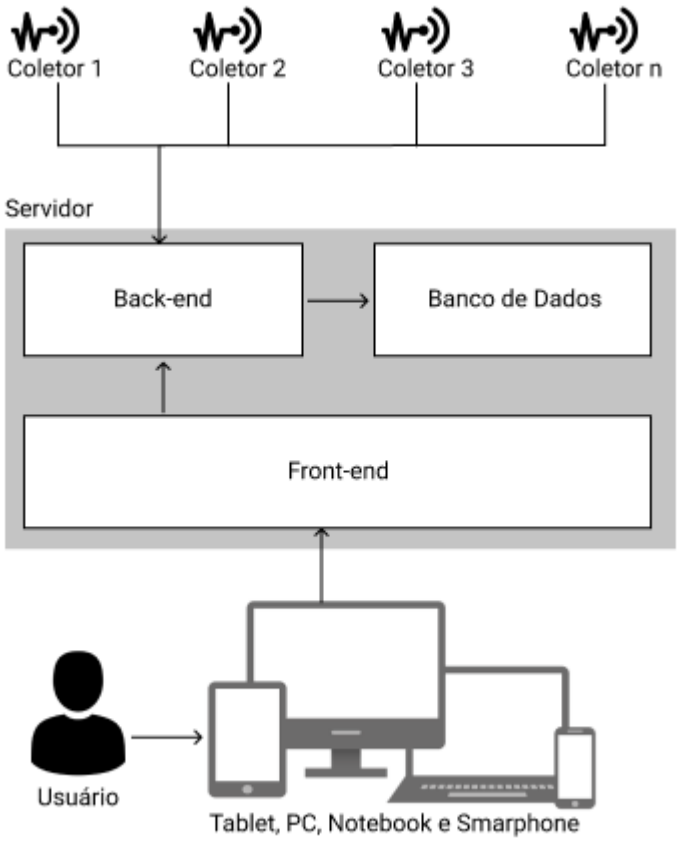
A partir da construção da fundamentação teórica e dos trabalhos relacionados, foi possível compreender inteiramente o escopo do MVP do projeto Renovar, que engloba um sistema distribuído de sensores que se comunicam com um servidor da Internet, por intermédio do protocolo HTTP, enviando dados de monitoramento ambiental.

Estes dados são armazenados no servidor e também disponibilizados visualmente, de forma geo localizada, através de uma interface web, portanto nesta proposta, foram utilizadas várias tecnologias, distribuídas em diversos escopos:

- Protocolos de comunicação, sensores, projeto de hardware;
- Persistência e processamento de dados;
- Visualização de dados geo localizados.

A Figura 10 representa o modelo da arquitetura de comunicação entre todos os componentes envolvidos no projeto: dispositivo de coleta de dados (coletores), servidor, *tablets*, *smartphones*, computadores tipo desktop e notebooks. Dentre os componentes apresentados é importante dividi-los em três grupos para compreender inteiramente a comunicação que é realizada: os coletores, o servidor e os clientes.

Figura 10 - Arquitetura de comunicação



Fonte: O autor (2018)

O primeiro grupo é composto pelos coletores, enumerados de 1 a “n”, que são os dispositivos de coleta que representam os nós de medição de qualidade do ar. Cada coletor é desenvolvido em parceria com o LCQAr, laboratório parceiro, que desde 2011 vem realizando pesquisas com o monitoramento de dados ambientais e criando variados dispositivos de coleta de material particulado (MP), CO, águas de chuvas, entre outros. Dessa forma, os coletores advindos dessa parceria aproveitam o arcabouço de dispositivos antigos aliados ao módulo de comunicação que foi desenvolvido para enviar os dados ao servidor, assim observa-se que um coletor é composto pelo seguintes módulos:

- O módulo de coleta, desenvolvido a partir da experiência e expertise do LCQAr, que possui os sensores para realizar as coletas de indicadores ambientais;
- O módulo de comunicação, que é composto por um sensor responsável por fazer conexão sem fio com a internet, realizando a comunicação com o

servidor. Este módulo foi construído pelo o autor do presente trabalho, agregando valor aos antigos dispositivos de coleta do LCQAr.

O funcionamento geral de um coletor inicia com a captura dos dados ambientais por meio do módulo de coleta, estes dados são transportados internamente para o módulo de comunicação. No módulo de comunicação os dados são estruturados no formato JSON e enviados para o servidor, utilizando um recurso da API REST disponibilizado pela aplicação back-end.

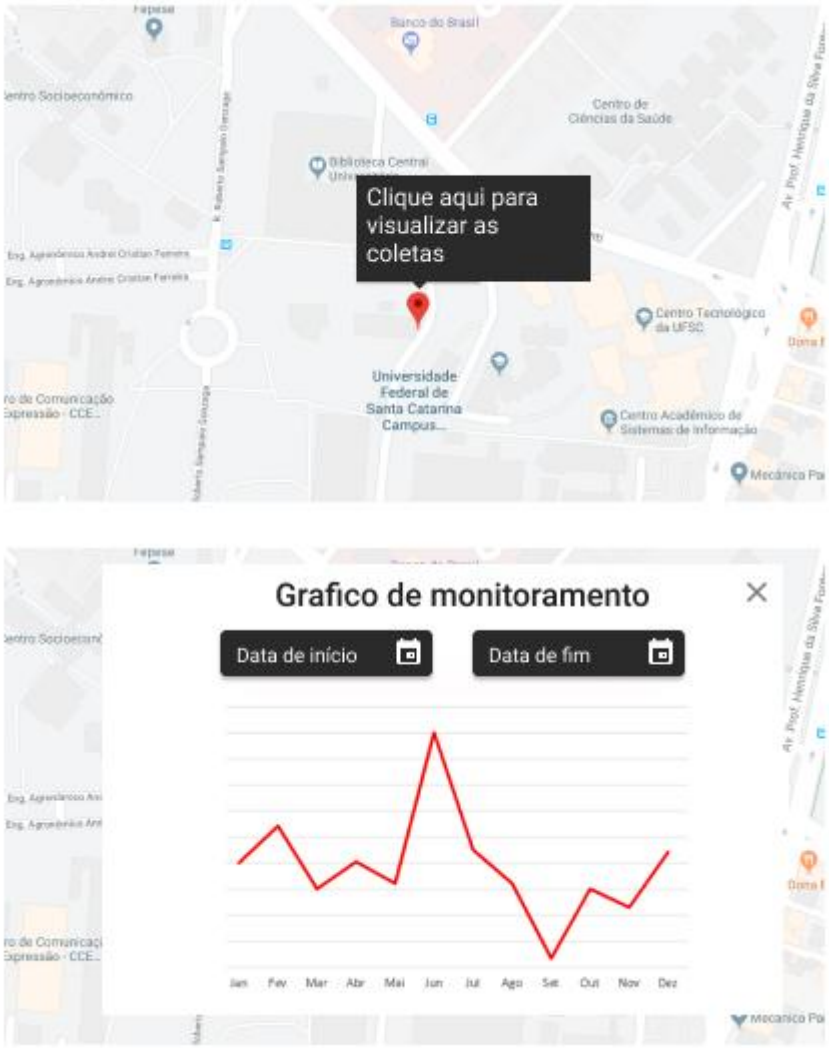
O segundo grupo possui fisicamente apenas um componente: o servidor, contudo, logicamente ele é formado por três principais entidades: aplicação back-end, banco de dados e a aplicação *front-end*.

- A aplicação back-end é o elo de comunicação entre os coletores e o banco de dados, também entre a aplicação front-end e o banco de dados. O maior processamento da aplicação está nesta entidade do sistema, pois é nela que são feitos os processos de seleção, tratamento, filtros e inserção dos dados. A aplicação back-end é responsável por disponibilizar uma API REST, que permite que os coletores enviem as informações de dados ambientais para o banco de dados, assim como a aplicação front-end consulte os dados pertinentes à exibição na interface web.
- O banco de dados é responsável por armazenar todas as informações referentes aos coletores e os seus respectivos sistemas de medição, as coletas realizadas por eles, assim como quais os indicadores que são monitorados (concentração de poluentes, temperatura, pressão e outros parâmetros). Somente a aplicação back-end tem acesso ao banco de dados, desta forma só é possível inserir, modificar, deletar e ler dados por meio da API REST.
- A aplicação Front-end é responsável pela comunicação dos usuários finais (clientes) com o servidor através de uma interface web exibindo dados geograficamente organizados. A aplicação front-end comunica-se com a aplicação back-end utilizando a API REST, a mesma utilizada pelos coletores. A interface web exibe os dados de monitoramento coletados, a série histórica, assim como permite o download de relatórios.

O terceiro grupo é composto por “n” componentes, os clientes, sendo eles qualquer um dos enumerados a seguir: computadores tipo desktop, notebook, *smartphone* ou *tablet*. Estes componentes estão nas mãos dos cidadãos,

os usuários da plataforma Renovar, que tem acesso aos dados da concentração dos poluentes atmosféricos medidos por cada coletor. Os dados são exibidos na interface web da aplicação *front-end*, em forma de gráficos acessados a partir de uma coordenada disponibilizada sob mapa, cmo exemplificado nos *wireframes* na Figura 11.

Figura 11 - Wireframes da aplicação Front-end



Fonte: O autor (2018)

A implementação da proposta da arquitetura é apresentada no capítulo seguinte, que descreve detalhadamente as etapas que contemplam o MVP da plataforma Renovar. De forma geral foi construído um dispositivo de monitoramento de baixo custo, instalado nas dependências do LCQAr, que coleta e mede a concentração de poluentes, e assim os envia para a aplicação back-end. Os dados

coletados são visualizados pela aplicação front-end, também detalhada nas próximas seções.

5 **PLATAFORMA RENOVAR**

A partir da definição da arquitetura apresentada no capítulo anterior, foi possível dar início ao processo de desenvolvimento do MVP da plataforma Renovar, dividido em três etapas: coletor, aplicação back-end e aplicação front-end. As decisões sobre ferramentas de implementação e tecnologias utilizadas foram tomadas a partir das experiências acadêmicas e profissionais do autor desse trabalho, aliadas ao auxílio de vídeo aulas para compreender alguma tecnologia específica e necessária durante o desenvolvimento do MVP.

Portanto, as tecnologias utilizadas nesta etapa do projeto envolvem as áreas de desenvolvimento orientado a objetos, desenvolvimento web, eletrônica, banco de dados relacionais, versionamento de código entre outras que implicam na construção de um MVP.

5.1 APLICAÇÃO BACK-END

O código da aplicação back-end pode ser visualizado na íntegra através do repositório do GitHub[3], assim como no apêndice A deste trabalho. De modo geral a aplicação back-end tem papel fundamental no projeto Renovar, pois é a única etapa independente de toda a plataforma. Tanto a aplicação front-end como o coletor dependem da implementação desta etapa do projeto, em função da API REST Renovar construída com ferramentas e tecnologias, que serão contempladas nas seções a seguir.

5.1.1 **Ferramentas utilizadas**

Para possibilitar produtividade de implementação e testes durante o desenvolvimento da API REST foram utilizadas as ferramentas listadas abaixo, elas são consideradas essenciais, pois oferecem a estrutura mínima para tornar o desenvolvimento possível.

- brModelo[4]: software utilizado para realizar a modelagem de banco de dados conceitual, lógico e físico;
- Java JDK[5]: em tradução direta é um Kit de Desenvolvimento Java, ou seja, um conjunto de ferramentas que permitem a construção de um sistema.

[3]Código fonte do projeto disponibilizado em: https://github.com/franciscosft/renovar
[4]Mais informações sobre o software brModelo acesse http://www.sis4.com/brModelo/

- Eclipse[6]: IDE escolhida para realizar a implementação da API;
- Postman[7]: software que constrói de forma rápida e fácil requisições HTTP;
- Git[8]: tecnologia utilizada para versionamento de código.

5.1.2 **Tecnologias utilizadas**

A API REST Renovar foi desenvolvida com as seguintes tecnologias: a linguagem de programação Java utilizando o *framework* Spring Boot e banco dados MySQL.

Os fatores que levaram a escolha dessas tecnologias para a implementação, além da proficiência do autor referente a linguagem de programação Java, o encapsulamento de recursos que o *framework* Spring Boot traz relacionados a infraestrutura de uma aplicação web, dentre eles podemos listar:

- Embarca um servidor: Tomcat, Jetty ou Undertow;
- Fornece dependências *starter* que simplificam a configuração de compilação;
- Não gera código sem necessidade;
- Não requer configuração de arquivos XML;

A aplicação back-end consiste na disponibilização de uma API para que tanto a aplicação front-end como os coletores tenham acesso aos recursos disponibilizados. Estes recursos tem acesso ao banco de dados da aplicação para inserção, leitura e atualização de informações.

5.1.3 **Implementação**

A implementação da API REST Renovar é dividida nos seguintes momentos:

a. etapa de configuração do ambiente;
b. modelagem de banco de dados;
c. desenvolvimento propriamente dito;

A etapa de configuração do ambiente, basicamente, envolve a instalação dos softwares listados na seção 5.1.1. Após a instalação, é importante adicionar o plugin

---

[5]Mais informações sobre Java JDK acesse https://www.oracle.com/technetwork/java/javase/overview/index.htm
[6]Mais informações sobre a IDE Eclipse acesse https://www.eclipse.org/downloads/packages/release/neon/2/eclipse-ide-java-developers
[7]Mais informações sobre o software Postman acesse https://www.getpostman.com/
[8]Mais informações sobre Git acesse https://git-scm.com/

Spring Tool Suite ao Eclipse. Ele auxilia no desenvolvimento e execução de aplicações construídas com Spring Boot. Também é necessário configurar o arquivo pom.xml, que gerencia as dependências de um projeto Maven para ficar semelhante ao do encontrado no apêndice A desse trabalho, nesse arquivo são declaradas todas as bibliotecas necessárias para a implementação de código.

Tendo em vista que o foco está na implementação, outras configurações adicionais serão comentadas no decorrer do texto, quando se fizer necessário. As seções seguintes referem-se a construção efetiva da aplicação back-end a partir da modelagem do banco de dados, ou seja, como foi o desenvolvimento de cada etapa, que tecnologias foram utilizada em cada momento e que arquitetura de implementação foi adotada.

5.1.4 **Modelagem de banco de dados**

A modelagem do banco de dados foi construída a partir de reuniões com a equipe do LCQAr, que deram os insumos necessários para compreender o escopo e as potencias entidades do modelo. As reuniões foram realizadas com professor Leonardo Hoinaski, coordenador do laboratório e coorientador do presente trabalho, os estudantes de graduação em Engenharia Sanitária e Ambiental Victoria Walendowsky e Igor Tibúrcio além do doutorando do programa de pós graduação de Engenharia Ambiental, Fernando Campos. Nas reuniões foram apresentados os projetos desenvolvidos pelo LCQAr abordando diversas frentes de pesquisa: MP e Indicadores de Saúde, emissão veicular, água das chuvas, queimadas.

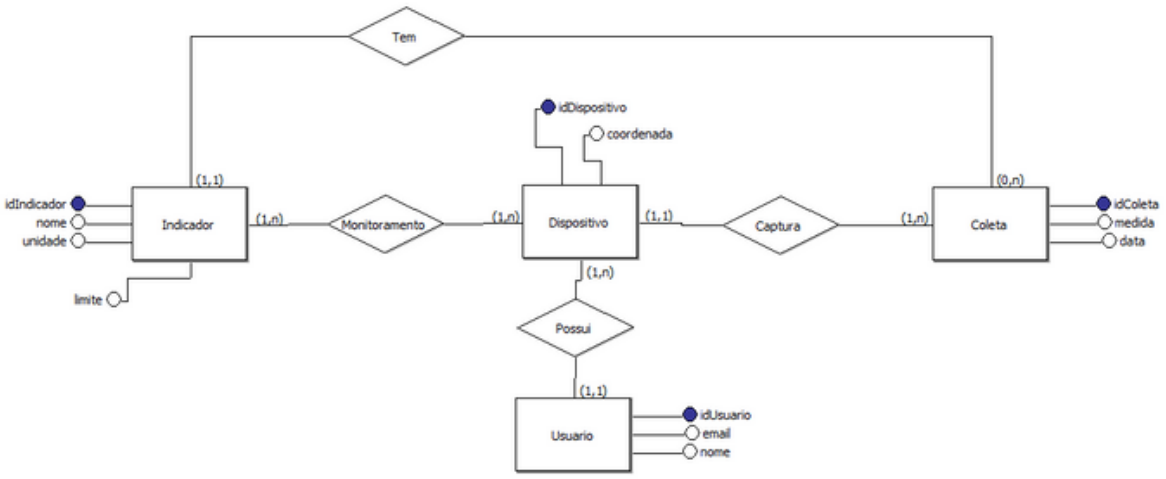
O professor e os estudantes enfatizam que o projeto MP e Indicadores de Saúde, cuja proposta de monitoramento de dados ambientais converge com a plataforma Renovar, tiveram como objetivo construir um dispositivo de coleta responsável por capturar o indicador MP, material particulado, armazenando os dados em um cartão SD, localizado no próprio dispositivo. Depois de um dia de capturas, o cartão SD é transportado para um computador para salvar os dados coletados.

5.1.4.1 Modelo conceitual

A compreensão do funcionamento do dispositivo de coleta foi essencial para o desenvolvimento de um modelo conceitual de banco de dados, pois foi possível

mapear as entidades e os relacionamentos envolvidos em todo o processo de coleta de MP. Observou-se que as entidades envolvidas nesse processo também podem ser aplicadas para qualquer outro tipo de indicador ambiental, conforme o diagrama de entidade e relacionamento exibido na Figura 12.

Figura 12 - Modelo entidade relacionamento



Fonte: O autor (2018)

Ao construir o modelo conceitual com a ferramenta brModelo, foram mapeadas quatro entidades: Indicador, Dispositivo, Coleta e Usuário, cada uma tem sua função, atributos e relacionamentos.

Indicador é a entidade responsável por representar os poluentes e as grandezas que podem ser monitoradas no meio ambiente pelo dispositivo. A relação entre Indicador e Dispositivo implica em uma cardinalidade N:N, ou seja, um indicador pode ser monitorado por nenhum ou vários dispositivos, assim como um dispositivo pode monitorar vários indicadores ou nenhum. Indicador também relaciona-se com a entidade Coleta implicando, desta vez em uma cardinalidade 1:N, isto significa que um indicador pode ter no mínimo uma e máxima N coletas, contudo uma coleta pode ter somente um indicador. É importante destacar que a entidade Indicador possui os seguintes atributos:

- Nome: representa o que está sendo monitorado ou seja, pressão, temperatura, CO2, MP, umidade.
- Unidade: representa o nome utilizado para medir uma determinada gradeza ou poluente, por exemplo: temperatura é medida em graus Celsius, pressão é medida em Pascal.

- Limite: representa o limiar que uma determinada grandeza ou poluente pode ter para não influenciar na saúde de uma pessoa. Tendo em vista que nem toda grandeza influencia diretamente na saúde de uma pessoa, este atributo é opcional.

A entidade Dispositivo é interpretada na prática pelos dispositivos de coleta. Relaciona-se com Indicador, Coleta e Usuário, ou seja é a principal entidade do modelo, pois é o elo entre todas. A relação entre a entidade Dispositivo/Indicador já foi explicada anteriormente, portanto é abordada, neste momento, a relação entre Dispositivo/Coleta e Dispositivo/Usuário. Dispositivo e Coleta possuem o mesmo tipo de relacionamento de Indicador e Coleta consequentemente a mesma cardinalidade, 1:N, neste caso uma determinada coleta pode ser capturada somente por um dispositivo, porém um dispositivo pode capturar no mínimo uma e máxima n coletas. Dispositivo e Usuário possuem também a cardinalidade 1:N, em que um usuário pode possuir no mínimo um e no máximo N dispositivos, mas um dispositivo possui no mínimo um e no máximo um usuário. O atributo relevante da entidade Dispositivo é a coordenada, composta por latitude e longitude, utilizada para indicar a sua localização.

A entidade Coleta tem como objetivo representar o histórico de capturas de um determinado dispositivo dado um determinado indicador. As duas relações dessa entidade já foram elucidadas, porém ainda é importante explicar quais são os atributos que a compõem:

- Medida: representa a quantidade de um indicador capturado pelo dispositivo.
- Data: representa a data de captura de uma determinada coleta.

Por fim ainda há a entidade Usuário que interpreta os participantes que desejam colaborar com o projeto Renovar. O participante é o usuário que quer contribuir construindo o seus dispositivos e enviar os dados capturados para o projeto. O usuário possui basicamente dois atributos: nome e e-mail.

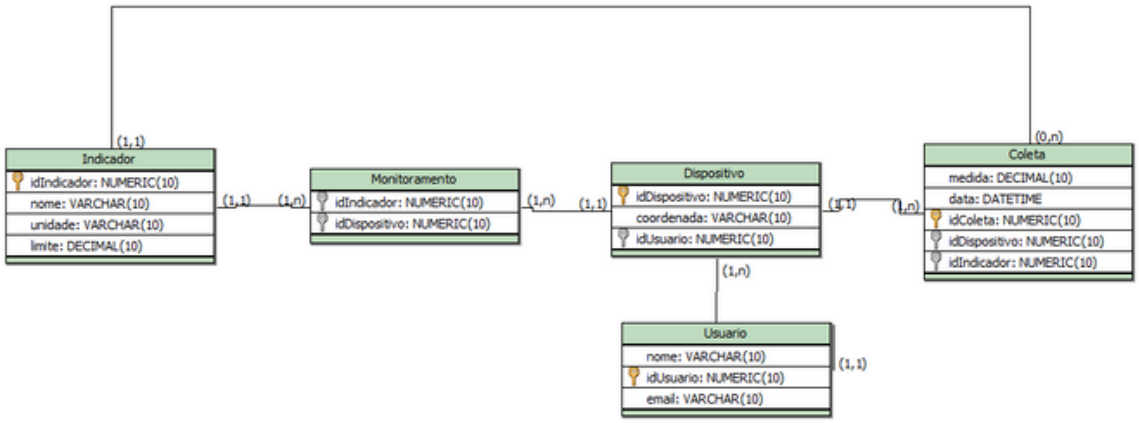
5.1.4.2 Modelo lógico

O modelo lógico foi construído facilmente a partir da ferramenta utilizada para desenvolver o modelo conceitual, o brModelo. De um modelo para outro as principais diferenças são:

- As entidades tornaram-se tabelas;
- Os atributos tornaram-se campos;
- Atributos principais, identificados por um círculo azul, são chamados de chaves privadas e representados pelo ícone chave na cor amarela;
- Um relacionamento de cardinalidade 1:N torna-se uma chave estrangeira, representada pelo ícone chave na cor cinza, na tabela que tem mais dependência;
- Um relacionamento de cardinalidade N:N torna-se uma nova tabela.

Conhecendo essas diferenças e analisando a Figura 13, é possível perceber que houve mudanças pontuais, relacionadas principalmente com a criação da tabela monitoramento e com o surgimento da chaves estrangeiras. Com a modelagem lógica definida é possível dar início ao desenvolvimento da API, pois durante o desenvolvimento dela que o banco de dados físico será construído com auxílio do Spring Boot.

Figura 13 - Modelo lógico



Fonte: O autor (2018)

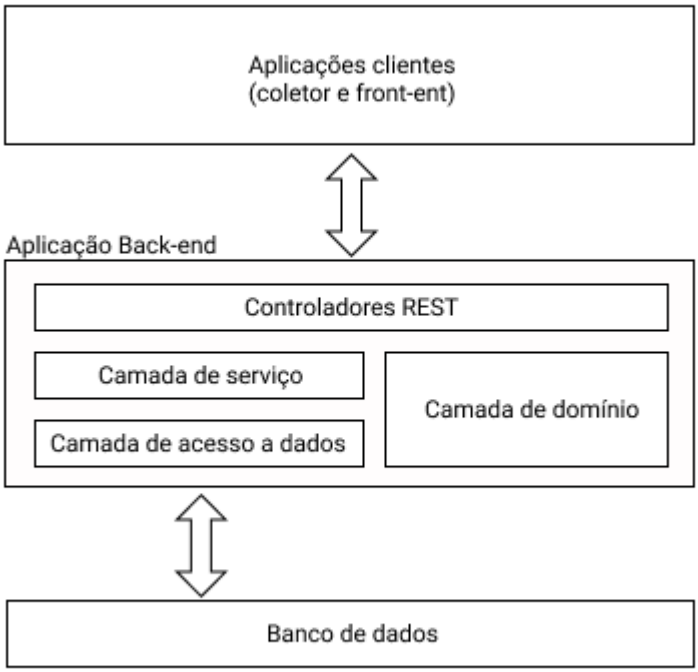
5.1.5 **API REST Renovar**

A API Renovar foi construída com a IDE Eclipse, plugin Spring Tool Suite e a linguagem de programação Java. Observa-se que para dar continuidade no desenvolvimento desta API em trabalhos futuros, é necessário o conhecimento prévio nas tecnologias citadas anteriormente ou em tecnologias equivalentes.

Para iniciar a implementação optou-se escolher o desenvolvimento em camadas que permite separar responsabilidades próprias de cada uma. Desta forma

a Figura 14 aumenta a especifidade da Figura 10, que permite visualizar com mais detalhes a aplicação back-end e a estrutura em camadas que está definida dentro dela.

Figura 14 - Estrutura em camadas



Fonte: O autor (2018)

Como percebe-se na Figura 14, a aplicação back-end é composta por quatro camadas: controladores REST, camada de serviço, camada de acesso a dados, e camada de domínio. Estas camadas desempenham funções específicas, e a partir dessa estrutura definida, precisa-se entender qual a responsabilidade de cada camada.

- Controladores REST é a camada de comunicação que permite aplicações clientes requisitem os recursos disponíveis e recebam as respostas necessárias. Nesta camada são estabelecidos os endpoints, messages bodys e header que cada recurso oferece;
- Camada de serviço está logo abaixo dos controladores REST, tem como objetivo intermediar a comunicação entre a camada de acesso aos dados e

os controladores REST, definindo as regras de negócio e as validações da aplicação.

- Camada de domínio reflete o modelo lógico de banco de dados, ou seja, nos lugares das tabelas há as classes que as representam. É importante destacar que a camada de domínio é conhecida por todas as outras, pois os objetos de domínio são utilizados em todas as partes da aplicação.
- Camada de acesso a dados comunica-se com o banco de dados da aplicação, somente ela além do administrador da aplicação possui acesso ao banco de dados.

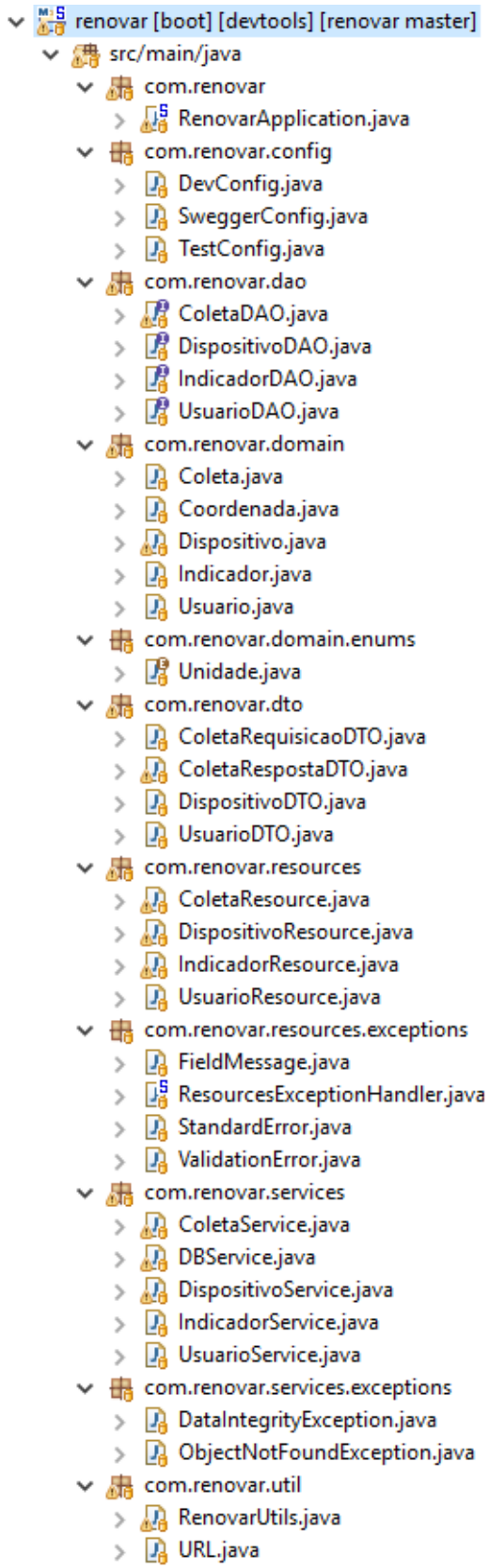
Com as responsabilidades compreendidas, as seções seguintes irão apresentar as implementações realizadas que correspondem a estrutura de camadas.

5.1.5.1 Visão Geral

A Figura 15 ilustra as classes criadas dentro dos seus pacotes. É preciso dar ênfase aos: “com.renovar.resoucers”, “com.renovar.services”, “com.renovar.domain” e “com.renovar.dao” que representam as camadas explicadas anteriormente: controladores REST, camada de serviço, camada de domínio e camada de acesso aos dados respectivamente.

Observa-se que nesses pacotes há classes cujo nomes possuem denominadores comuns, ou seja, em todos os pacotes há classes que começam com “Usuário”, “Indicador”, “Coleta” e “Dispositivo”. Elas têm estruturas de programação bastante similares, então para evitar redundância de explicações foram utilizadas como exemplo as classes que possuem “Coleta” como sufixo, uma vez que possuem código eventualmente mais elaborado.

Figura 15 - Visão geral da aplicação back-end



Fonte: O autor (2018)

5.1.5.2 Camada de Domínio

A camada de domínio reflete o banco de dados da aplicação, desta forma as classes presentes nessa camada são tabelas no banco de dados físico. Na Figura 16 observa-se trecho de código da classe “Coleta”, em que é declarado o nome da classe e os seus campos. Para cada campo declarado são criados métodos “getters” e “setters”. Os métodos “getters” são estruturas de códigos responsáveis por buscar o valor de um determinado campo, já os métodos “setters” são estruturas que realizam a atribuição de um valor para um campo.

Figura 16 - Coleta

```
16  @Entity
17  public class Coleta implements Serializable {
18
19      private static final long serialVersionUID = 1L;
20
21      @Id
22      @GeneratedValue(strategy = GenerationType.IDENTITY)
23      private Integer id;
24      private Double medida;
25      @JsonFormat(pattern = "dd/MM/yyyy HH:mm:ss")
26      private Date data;
27      private Double latitude;
28      private Double longitude;
29
30      @JsonIgnore // utilizado para esconder os objetos Json
31      @ManyToOne
32      @JoinColumn(name = "dispositivo_id")
33      private Dispositivo dispositivo;
34
35      @ManyToOne
36      @JoinColumn(name = "indicador_id")
37      private Indicador indicador;
38
39      public Coleta() {
40      }
```

Fonte: O autor (2018)

O autor fez uso do framework Hibernate, criado com o objetivo de auxiliar desenvolvedores a abstrair diferenças entre o paradigma orientado a objetos e o modelo relacional de banco de dados. O Hibernate possui o pacote “javax.persistence” que implementa as notações utilizadas na figura 16: “@Entity”, “@JoinColum”, “@ManyToOne”, “@Id” e “@GeneratedValue”.

- Entity: notação utilizada na linha 16 do trecho de código da figura para conectar classe à tabela, ou seja, no banco dados há uma tabela com o mesmo nome da classe. Dessa forma objetos do tipo “Coleta” podem ser persistidos na tabela Coleta.

- JoinColum: notação responsável por nomear a coluna que possui chave estrangeira e relacionar com o campo pelo qual é representado declarado na classe. As linhas 32 e 36 exemplificam isso, elas fazem o uso da notação, na primeira ocorrência a coluna “dispositivo_id” chave estrangeira da tabela Coleta é representado pelo campo “dispositivo”, a segunda ocorrência segue a mesma ideia.
- ManyToOne: notação responsável por apresentar o tipo relacionamento entre as classes da mesma forma que é feito no banco de dados. Neste caso um objeto da classe Coleta pode estar relacionado apenas um objeto da classe Dispositivo e Indicador;
- Id: notação visualizada na linha 21, responsável por identificar o campo que representa a chave primária no banco de dados.
- GenaratedValue: notação cuja função é gerenciar a criação de identificador único, toda vez que um novo objeto do tipo “Coleta” for persistido no banco de dados. É importante destacar que essa notação deve estar logo após da notação “@Id”.

No trecho de código da Figura 16, ainda há duas notações “@JsonIgnore” e “@JsonFormat”. A primeira é responsável por esconder os dados do campo “indicador” quando for impresso um objeto da classe “Coleta”, já a segunda notação exige que valores que sejam atribuídos ao campo “data” estejam formatados conforme a linha 25. Também é preciso dar enfase que a classe em questão implementa a interface “Serializable”, isto significa que um objeto do tipo “Coleta” pode ser transformado em uma sequências de bytes e enviado pela internet, assim como salvo em disco e até mesmo enviado para outra JVM.

É importante ressaltar que as demais classes dessa camada seguem o mesmo padrão da exibida no trecho de código da Figura 16. Observa-se que os atributos das tabelas apresentadas na Figura 13 tornam-se os campos das classes, o que permite compreender como elas estão estruturadas na camada de domínio.

5.1.5.3 Camada de acesso a dados

Após o entendimento da camada de domínio é importante dar sequências as explanações com a camada de acesso aos dados, pois essa camada é responsável

pelas operações de inserção, deleção, atualização e leitura que a aplicação faz ao banco de dados.

O framework Spring Boot permite que a implementação dessa camada seja bastante simples, dessa forma as interfaces que correspondem ao acesso a dados precisam apenas estender a classe “JpaRepository”, incluindo os tipo da entidade e identificador. O trecho de código pode ser visualizado Figura 17.

Figura 17 - Declaração da interface ColetaDAO

```
16
17 @Repository
18 public interface ColetaDAO extends JpaRepository<Coleta, Integer> {
19
```

Fonte: O autor (2018)

Em algumas interfaces como “IndicadorDAO”, que pode ser consultada no apêndice A, não houve necessidade de fazer mais nenhuma implementação, isto porque a classe “JpaRepository” encapsula uma série de métodos, que serão vistos na seção seguinte, que fazem as operações no banco de dados.

Contudo, há momentos que é inevitável fazer uma operação específica, neste cenário é possível utilizar a notação “@Query”, que por sua vez faz uso de JPQL, um tipo de linguagem para criar consultas. As linhas 38 e 39 do trecho de código da Figura 18 podem exemplificar o uso da notação para realizar uma operação de seleção no banco de dados. Neste caso a operação busca todas as coletas cujo identificador do dispositivo e indicador sejam iguais as variáveis “idDispotivo” e “idIndicador” passadas como parâmetro e anotadas com “@Param”, e que também estejam dentro de um intervalo definido pelas variáveis “dataInicio” e “dataFim” também passadas como parâmetro e anotadas com “@Param”. Outro aspecto do código que merece destaque é a propriedade “readOnly” definida pela notação “@Transactional” na linha 37 do trecho de código da figura 18, que tem o objetivo de impedir que a consulta realize qualquer tipo de operação de escrita na banco de dados, possibilitando apenas a operação de leitura.

Figura 18 - Exemplo de uso de @Query

```
37⊖   @Transactional(readOnly = true)
38    @Query("SELECT obj FROM Coleta obj WHERE obj.dispositivo.id = :idDispositivo AND obj.indicador.id = :idIndicador AND obj.data BETWEEN :dataInicio"
39          + " AND :dataFim ORDER BY obj.data ASC")
40    List<Coleta> buscarColetasDispositivoIndicadorData(@Param("idDispositivo") Integer idDispositivo,
41          @Param("idIndicador") Integer idIndicador, @Param("dataInicio") Date dataInicio,
42          @Param("dataFim") Date dataFim);
43
```

Fonte: O autor (2018)

5.1.5.4 Camada de serviço

Outra funcionalidade que o Spring Boot fornece e que camada de serviço vai apresentar é a notação “@Autowired”, responsável por fazer injeção de dependências, ou seja, declarar uma classe em escopo global sem a necessidade de instanciá-la.

As injeções de dependências começam a ser utilizadas na camada de serviço, pois elas implementam as regras de negócio e validações necessitando acesso a camada de dados para realizar essas operações. No trecho de código da Figura 19 a classe “ColetaService” mostra a injeção de “ColetaDAO” nas linhas 35 e 36, uma vez realizada a injeção é possível utilizar o objeto definido quantas vezes for necessário. Na classe “ColetaService” também são injetadas outras dependências, a classe “DispositivoService” e a “IndicadorService”, responsáveis por acessar os dados das tabela Dispositivo e Indicador respectivamente.

Figura 19 - Camada de serviço

```
30
31 @Service
32 public class ColetaService {
33     private final Logger log = LoggerFactory.getLogger(ColetaService.class);
34
35     @Autowired
36     private ColetaDAO dao;
37
38     @Autowired
39     private DispositivoService dispositivoService;
40
41     @Autowired
42     private IndicadorService indicadorService;
43
44     public Coleta buscarColeta(Integer idColeta) {
45         Optional<Coleta> findById = dao.findById(idColeta);
46         return findById.orElseThrow(() -> new ObjectNotFoundException(
47                 "Coleta não encontrada: " + idColeta + " , Tipo: " + Coleta.class.getName()));
48     }
49
```

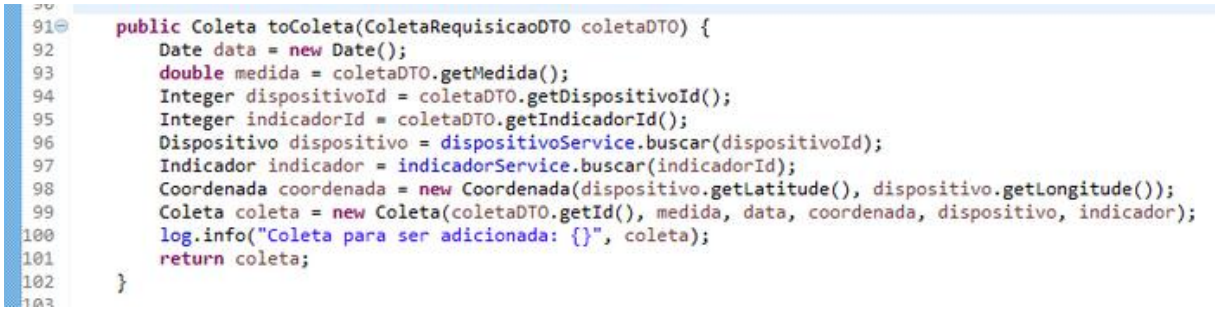
Fonte: O autor (2018)

A injeção da dependência “ColetaDAO” é utilizada na linha 45 do trecho de código da Figura 19 para buscar uma coleta dado um determinado identificador. O método em questão utilizado “findById” não é definido explicitamente na interface “ColetaDAO”, pois ele está encapsulado na classe que a interface estende: “JpaRepository”.

O trecho de código da figura 20 é bastante interessante e merece alguns comentários, pois observa-se o uso das dependências “DispositivoService” e

“IndicadorService”, além de trazer uma classe que ainda não apresentada: “ColetaRequisicaoDTO”.

Figura 20 - Método toColeta



```
91  public Coleta toColeta(ColetaRequisicaoDTO coletaDTO) {
92      Date data = new Date();
93      double medida = coletaDTO.getMedida();
94      Integer dispositivoId = coletaDTO.getDispositivoId();
95      Integer indicadorId = coletaDTO.getIndicadorId();
96      Dispositivo dispositivo = dispositivoService.buscar(dispositivoId);
97      Indicador indicador = indicadorService.buscar(indicadorId);
98      Coordenada coordenada = new Coordenada(dispositivo.getLatitude(), dispositivo.getLongitude());
99      Coleta coleta = new Coleta(coletaDTO.getId(), medida, data, coordenada, dispositivo, indicador);
100     log.info("Coleta para ser adicionada: {}", coleta);
101     return coleta;
102 }
```

Fonte: O autor (2018)

Para explicar o método “toColeta”, primeiro é necessário definir o que é um DTO, em tradução direta é um objeto de transferência de dados, ou seja é um objeto utilizado para comunicação entre sistemas diferentes, que precisam conhecer a mesma estrutura de dados. Os objetos DTOs representam de forma mais simplificada os objetos da camada de domínio, portanto eles abstraem informações que não se fazem necessárias. Neste caso a classe “ColetaRequisicaoDTO” foi criada para facilitar o envio de dados originados do dispositivo de coleta, assim ele é composto somente pelos campos essências para a realização da coleta: identificador do dispositivo, indicador e o valor da medida. Além desse DTO foram implementados outros que estão disponíveis no apêndice A.

Com o entendimento do conceito de DTO é possível explicar o método “toColeta”, que tem a função de traduzir um objeto “ColetaRequisicaoDTO” para um objeto “Coleta”. Inicialmente na linha 92 é instanciado a variável “data”, que representa a data de captura de uma determinada grandeza ou poluente, em seguida são extraídos os dados essências da instância “coletaDTO”, visualizados nas linhas 94, 95 e 96. Nas linhas 97 e 98 realiza-se a busca do dispositivo e indicador dado o seu identificador, essa busca é implementada de forma semelhante ao trecho de código da Figura 19. Por fim é construído um objeto do tipo “Coleta” com as informações provenientes da instância “coletaDTO” aliadas as regras de negócio da camada de serviço.

5.1.5.5 Controladores REST

Para finalizar as explicações sobre as camadas que compõem a aplicação back-end, nesta seção é apresentado os controladores REST, camada que representa uma interface de entrada aos recursos da aplicação. Os controladores REST acessam o banco de dados por intermédio das camadas de serviço e acesso aos dados.

O Spring Boot auxilia ao criar um controlador REST, pois ele fornece as notações “@RestController” e “@RequestMapping” que são utilizadas imediatamente antes de declarar uma classe, conforme visualizado nas linhas 33 e 34 do trecho de código da Figura 21. A notação “@RestController” tem a função de atribuir a classe que esta utilizando características REST que permite manipular as requisições vindas de clientes. Já a notação “@RequestMapping” informa qual deve ser o caminho de acesso ao controlador REST, neste caso todas as requisições para “ColetaResource” terão como base a URL “localhost:8080/coletas”. Também é importante observar que na linha 39, do trecho de código da Figura 21, está sendo injetado a dependência “ColetaService”. Ela é utilizada em todos os métodos da classe “ColetaResource”, pois é responsável por fazer a comunicação entre a camada de serviço e os controladores REST.

Figura 21 - Declaração ColetaResource

```
33 @RestController
34 @RequestMapping(value = "/coletas")
35 public class ColetaResource {
36     private final Logger log = LoggerFactory.getLogger(ColetaResource.class);
37
38     @Autowired
39     private ColetaService service;
40
```

Fonte: O autor (2018)

O trecho de código da Figura 22 apresenta o método “buscarIndicadoresDatas” que é anotado com “@CrossOrigin” e também com “@RequestMapping”. A notação “@CrossOrigin” é utilizada para permitir que requisições do tipo GET de qualquer origem acessem a URL “localhost:8080/coletas/intervalo”. A partir disso já é possível compreender a função da notação “@RequestMapping” ao ser utilizada na declaração de um método, ela determina o tipo de verbo HTTP e o endpoint que uma requisição deve ter para acessar o recurso.

Outra notação que ainda não fora apresentada é “@RequestParam”, que diz quais parâmetros podem ser passados a uma requisição, ela é utilizada das linhas 70 a 73 do trecho de código da Figura 22. Com o entendimento das notações percebe-se que o método apresentado, no trecho de código da Figura 22, tem a função de buscar uma lista de coletas dado um identificador de dispositivo, indicador e intervalo de datas que são passados como parâmetro através da notação “@RequestParam”.

Figura 22 - Método para buscar Coleta

```
67    @CrossOrigin
68    @RequestMapping(value = "/intervalo/", method = RequestMethod.GET)
69    public ResponseEntity<List<ColetaRespostaDTO>> buscarIndicadoresDatas(
70            @RequestParam(value = "idDispositivo", defaultValue = "") Integer idDispositivo,
71            @RequestParam(value = "idIndicador", defaultValue = "") Integer idIndicador,
72            @RequestParam(value = "dataInicio", defaultValue = "") String inicio,
73            @RequestParam(value = "dataFim", defaultValue = "") String fim) {
74
75        if(inicio.isEmpty() || fim.isEmpty()) {
76            return ResponseEntity.badRequest().body(null);
77        }
78
79        Date dataInicio2 = RenovarUtils.toDataInicio(inicio);
80        Date dataFim2 = RenovarUtils.toDataFim(fim);
81        log.info("Buscando coletas entre {} e {}", dataInicio2, dataFim2);
82
83        List<Coleta> coletas = service.buscarColetasDispositivoIndicadorData(idDispositivo, idIndicador, dataInicio2, dataFim2);
84        List<ColetaRespostaDTO> collect = coletas.stream().map(c -> new ColetaRespostaDTO(c))
85                .collect(Collectors.toList());
86        return ResponseEntity.ok().body(collect);
87    }
```

Fonte: O autor (2018)

Um método que merece destaque pode ser visualizado no trecho de código da Figura 23, que é responsável por inserir uma coleta no banco de dados. Neste caso observa-se o uso da notação “@RequestBody” na linha 129. Esta notação informa o message body que será aceito pela API ao ser realizado uma requisição POST na URL “locahost:8080/coletas”. Neste caso o message body deve seguir o padrão estabelecido pela classe “ColetaRequisicaoDTO”, caso contrário a API REST irá recusar a requisição .

Figura 23 - Inserir coleta

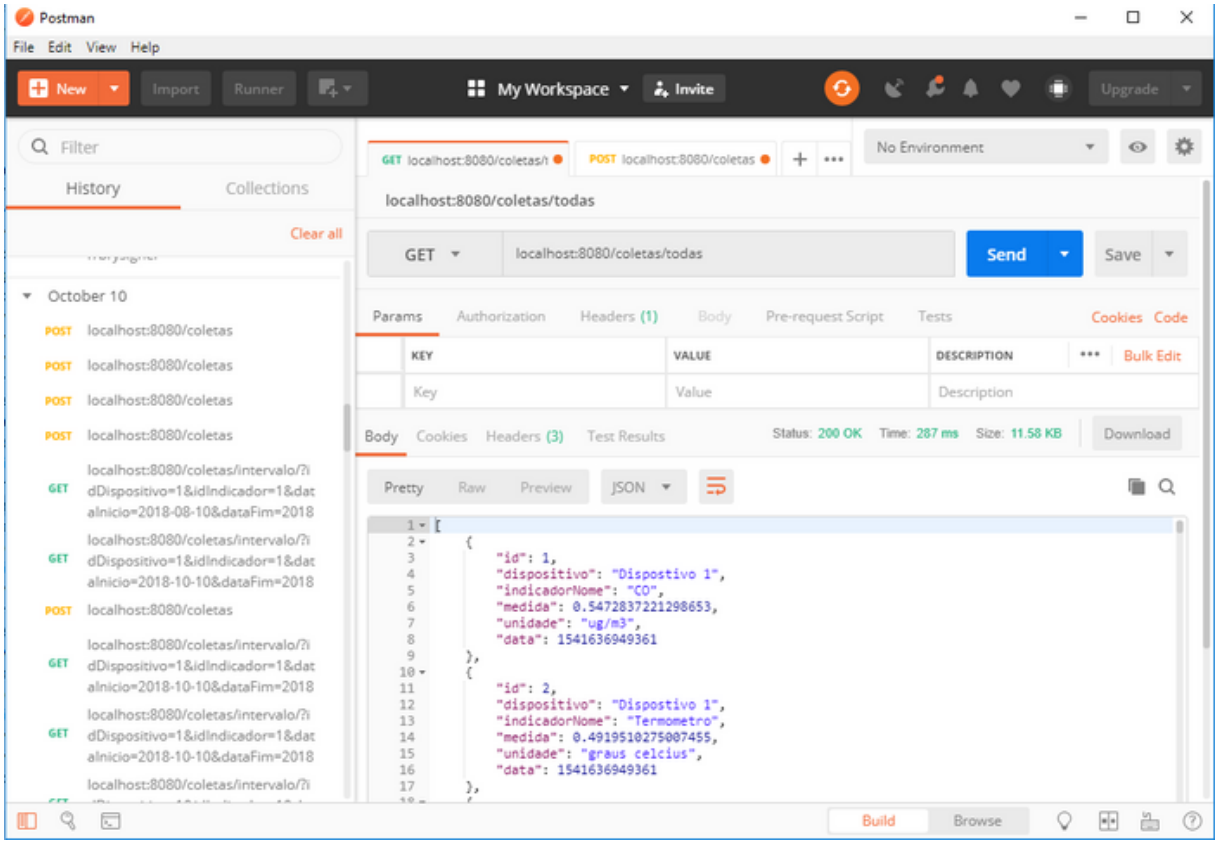
```
127    @ApiOperation(value = "Método utilizado para adicionar uma coleta")
128    @RequestMapping(method = RequestMethod.POST)
129    public ResponseEntity<Void> cadastrarColeta(@RequestBody ColetaRequisicaoDTO coletaDTO) {
130        Coleta coleta = service.toColeta(coletaDTO);
131        coleta = service.inserir(coleta);
132        URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(coleta.getId()).toUri();
133        return ResponseEntity.created(uri).build();
134    }
```

Fonte: O autor (2018)

No processo de implementação da API REST Renovar foi utilizado o software Postman para realização de testes, a fim de verificar o funcionamento desta. Com o

Postman é possível realizar requisições a aplicação back-end a partir dos endpoints definidos na camada de controladores REST. Na Figura 24 observa-se um requisição do tipo GET feita ao endpoint “localhost:8080/coletas/todas” que retorna uma lista de todas as coletas realizas.

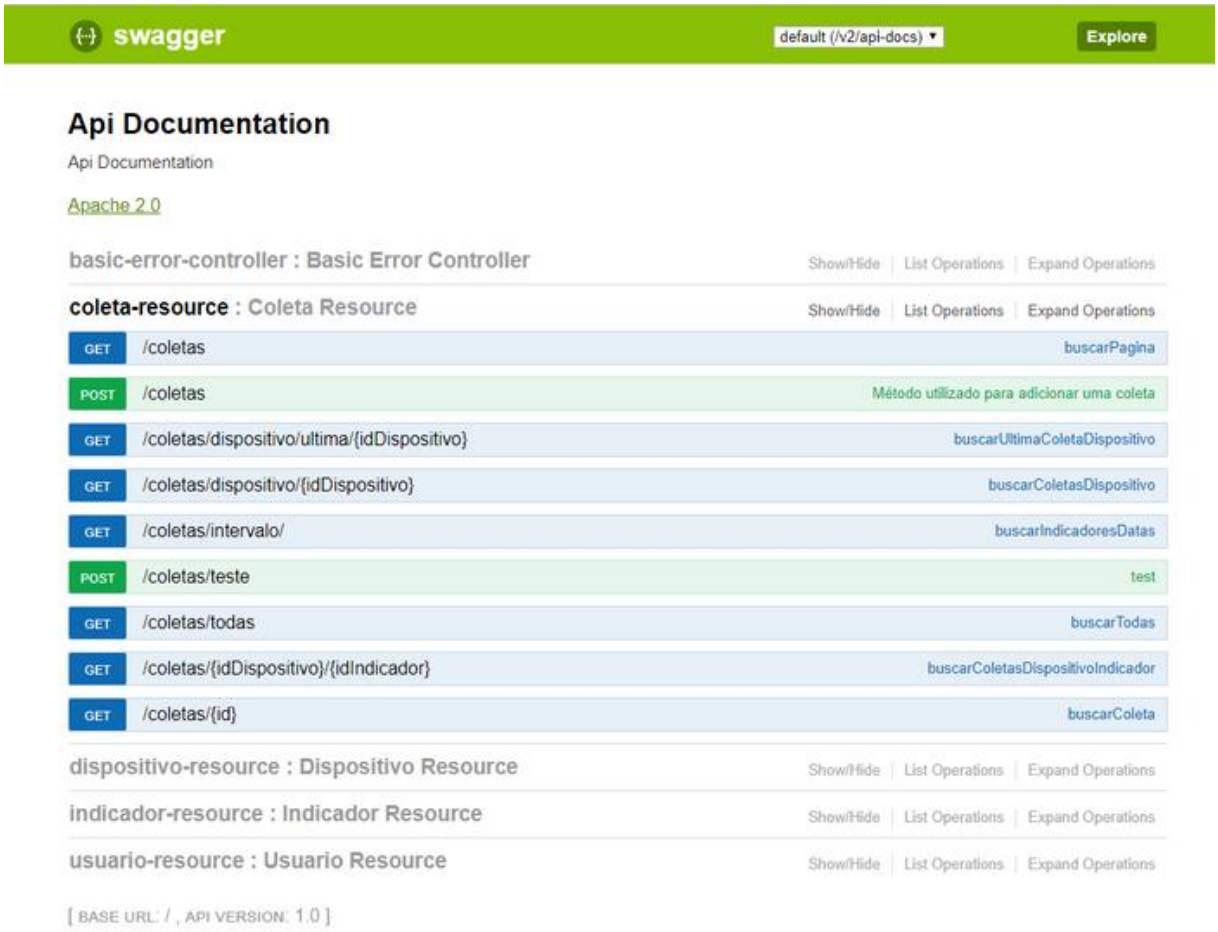
Figura 24 - Postman



Fonte: O autor (2018)

Assim como o Postman, o Swagger poder ser utilizado para verificar o funcionamento da aplicação back-end, contudo ele também é uma ferramenta que é bastante utilizada para documentar uma API REST. Desta forma o Swagger disponibiliza acesso aos endpoints de uma API, que tipo de verbo é utilizado em cada um, quais parâmetros e objetos devem ser utilizados. Para isso é necessário realizar a configuração do Swagger, disponível no apêndice A do presente trabalho, após já é possível visualizar os endpoints da API REST conforme Figura 25.

Figura 25 - Swagger



Fonte: O autor (2018)

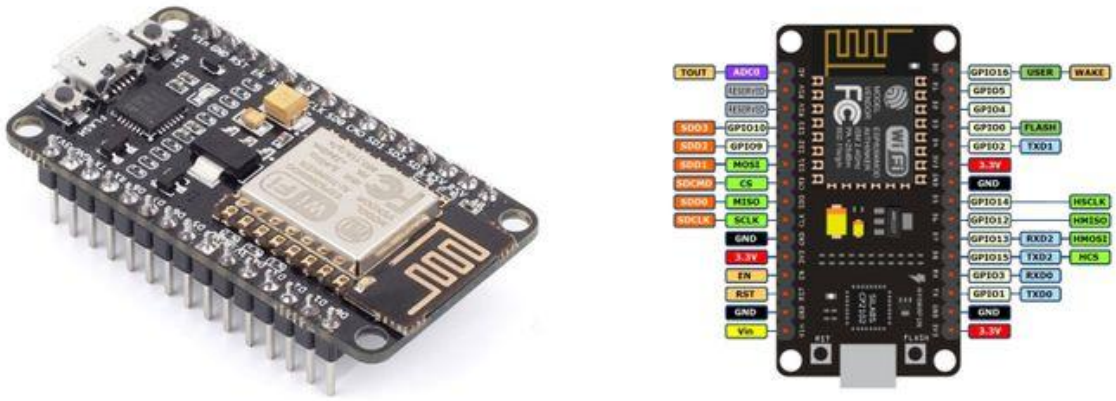
5.2 COLETOR

O coletor, descrito com maiores detalhes no anexo A, é um dispositivo construído pelo LCQAr com a placa NodeMCU. Acoplado a ela há os seguintes sensores: temperatura, pressão, concentração de CO e umidade relativa. A plataforma Renovar permite que os coletores sejam flexíveis em relação aos sensores acoplados, ou seja, é possível remover ou adicionar novos sensores de acordo com a necessidade de monitoramento. É importante ressaltar que o presente trabalho não contempla o desenvolvimento integral do coletor conforme elucidado na Introdução seção 1.5. Contudo, foi necessário desenvolver o módulo de comunicação sem fio, contribuindo com o projeto do LCQAr.

5.2.1 **Dispositivo**

Para o desenvolvimento de comunicação sem fio entre coletor e aplicação back-end, foi utilizado o módulo Wifi ESP8266 NodeMCU, que é baseado em um modelo de open source de hardware, semelhante a proposta do Arduino. Como o dispositivo está combinado com o chip ESP8266 é possível criar uma variedade de aplicações, com conexão sem fio a Internet ou Intranet. O NodeMCU também possui um conversor TTL-Serial e um regulador de tensão de 3.3V, além de ser facilmente acoplado a protoboard (THOMSEM, 2016).

Figura 26 - Módulo NodeMCU ESP8266



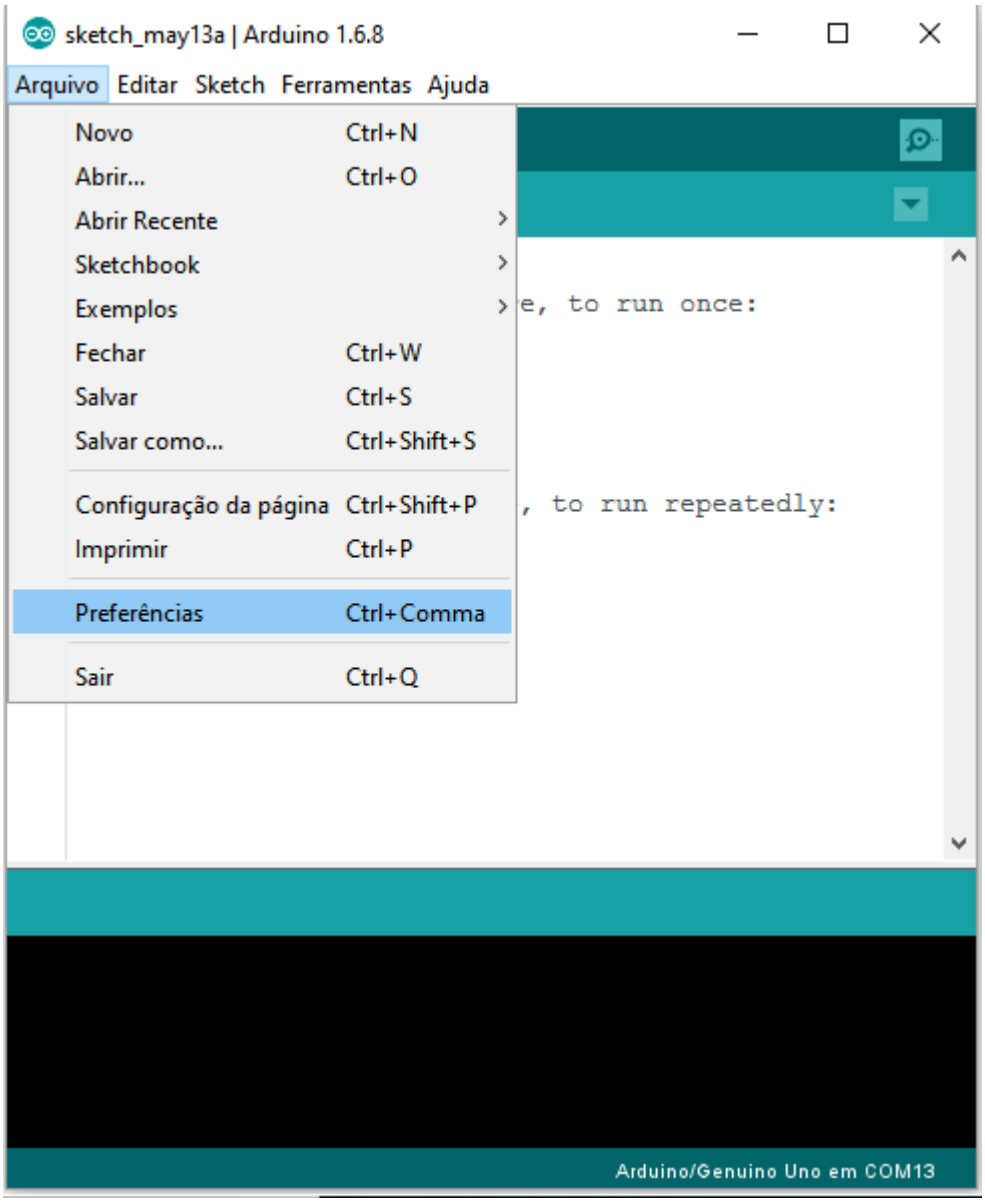
Fonte: Thomsem (2016)

5.2.2 **Plataforma de implementação**

O NodeMCU pode ser programado utilizando a linguagem de programação Lua, ou ainda a IDE Arduino[9], que no caso foi a escolhida para a implementação em função da experiência do autor. Thomsen (2016) explica quais são as configurações adicionais necessárias para programar NodeMCU com a IDE Arduino. Primeiramente é necessário abrir o programa, visualizar a barra de ações, acessar a opção “Arquivo” e selecionar “Preferências”, conforme Figura 27.

[9]A IDE Arduino pode ser encontrada em https://www.arduino.cc/en/Main/Software#, acessado em 22 de outubro de 2018

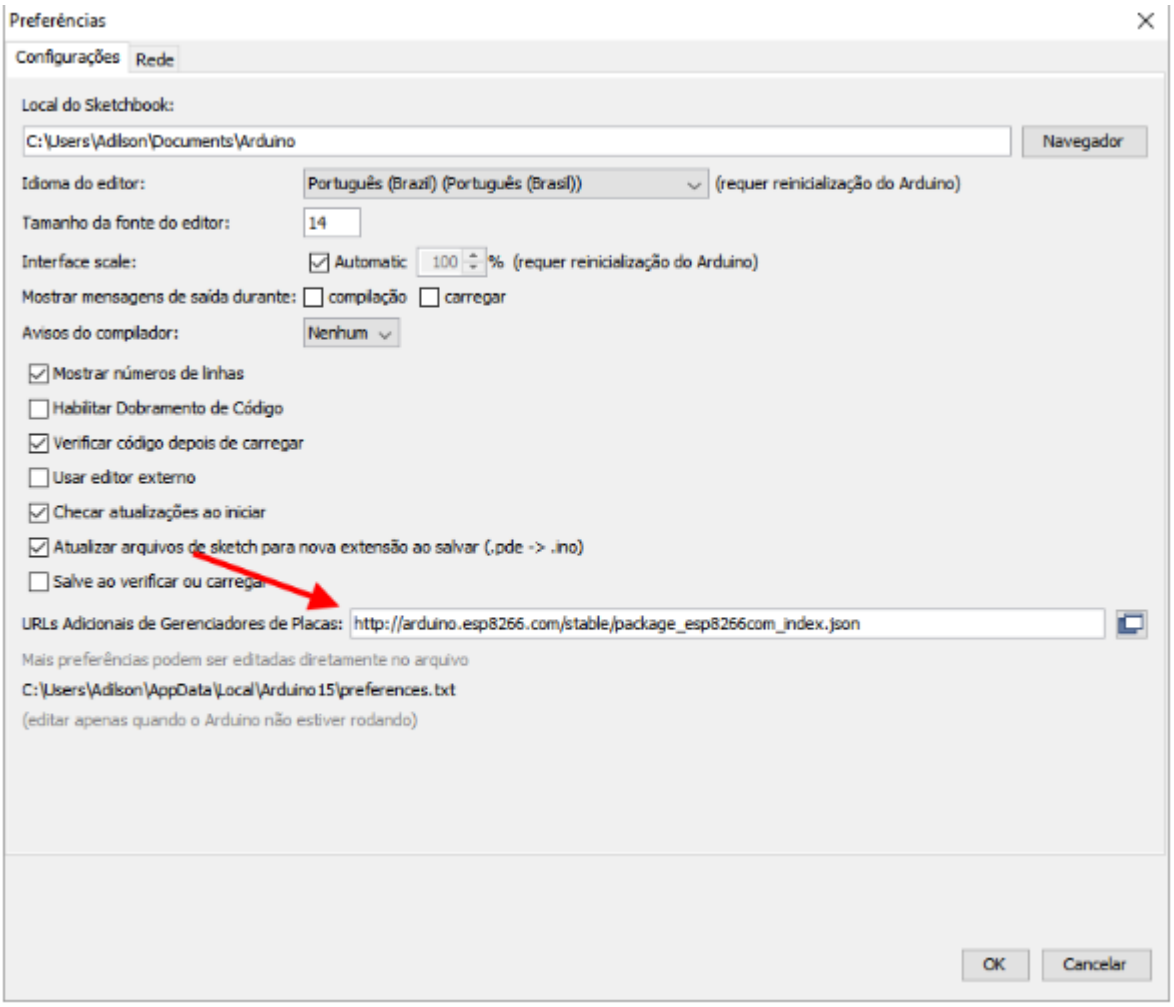
Figura 27 - Configuração IDE Arduino



Fonte: Thomsen (2016)

Será aberto uma tela de configurações, Figura 28, onde é possível escolher as preferências do sistema. Dentre elas é necessário preencher o campo "URLs adicionais de Gerenciadores de Placas" com o seguinte *link*: "http://arduino.esp8266.com/stable/package_esp8266com_index.json", que é responsável por disponibilizar acesso a download para a etapas subsequente. Após, clicar no botão “OK” para voltar a tela principal.

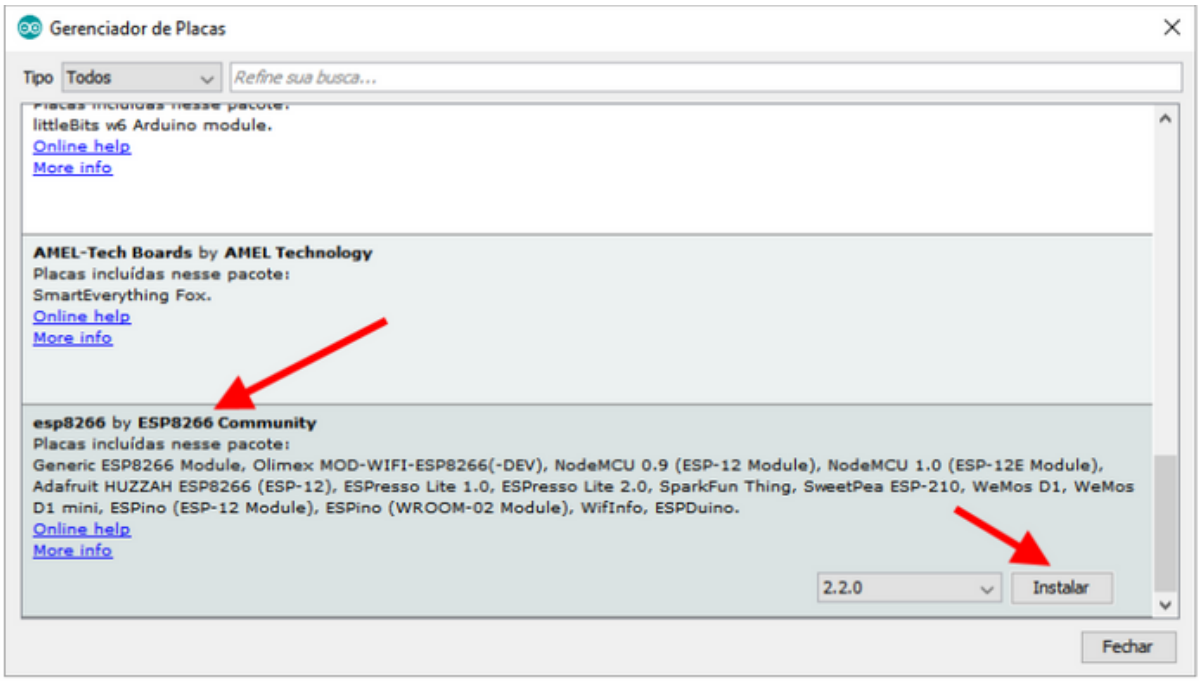
Figura 28 - Configuração da IDE Arduino



Fonte: Thomsen (2016)

A próxima etapa de configuração da plataforma envolve selecionar a placa que será utilizada para efetiva programação. Para isso é preciso clicar na opção "Ferramentas" na tela principal, em seguida expandir o menu "Placa: Arduino/Genuino Uno" e selecionar o item "Gerenciador de Placas". De acordo com a Figura 29 é necessário procurar por "esp8266 by ESP8266 Community" e, ao encontrar clicar em "Instalar".

Figura 29 - Gerenciador de placas



Fonte: Thomsen (2016)

Ao finalizar a instalação é necessário abrir, novamente, o gerenciador de placas e selecionar a opção NodeMCU 1.0 (ESP-12E Module). Neste momento as configurações da plataforma estão prontas, já é possível começar a programar o NodeMCU.

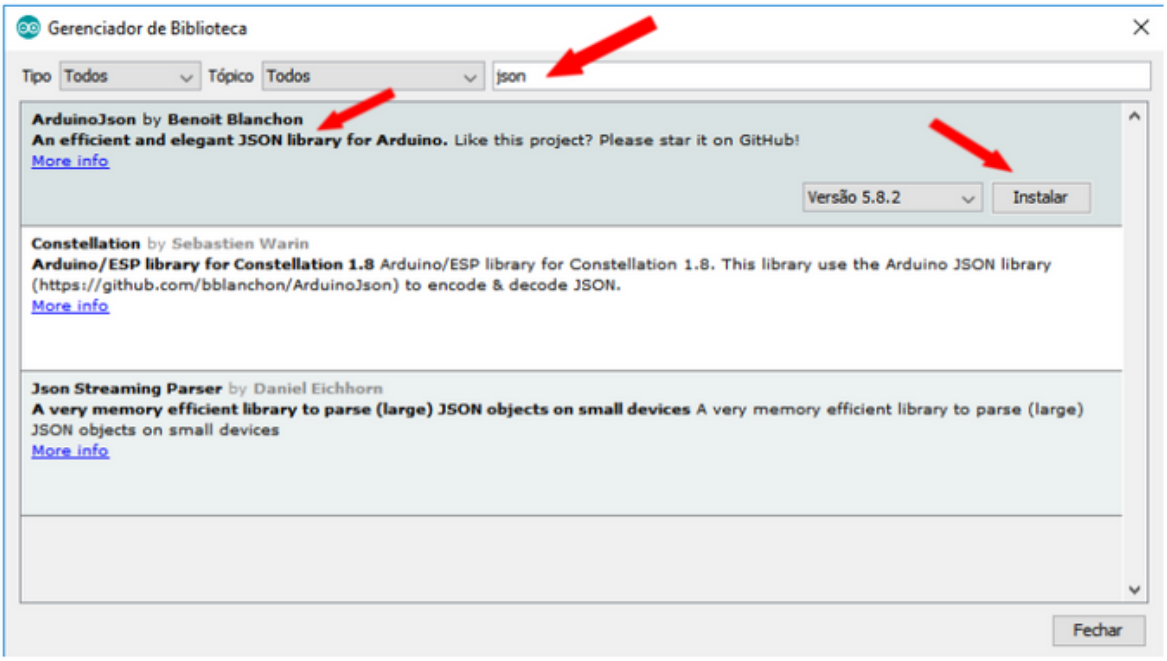
5.2.3 **Implementação do módulo de comunicação**

Nesta seção é explicado como funciona o módulo de comunicação implementado para o dispositivo de coleta, com código disponibilizado na íntegra no apêndice B deste trabalho, abstraindo a integração com demais sensores. A comunicação do coletor foi desenvolvida com base em Bauermeister (2017), que explica como utilizar a biblioteca “ArduinoJSON” para montar uma requisição do tipo POST, conforme o protocolo HTTP. Para isso, segundo Bauermeister (2017), é necessário importá-la seguindo os seguintes passos:

- Selecionar a opção "Sketch" disponível na tela principal da IDE Arduino;
- Expandir o item "Incluir Biblioteca";
- Clicar em "Gerenciar Bibliotecas".

Após é possível visualizar a Figura 30, que exibe a tela que será aberta ao realizar o último passo da etapa anterior. Nesta tela é preciso procurar pela biblioteca "ArduinoJson by Benoit Blanchon" e realizar a sua instalação.

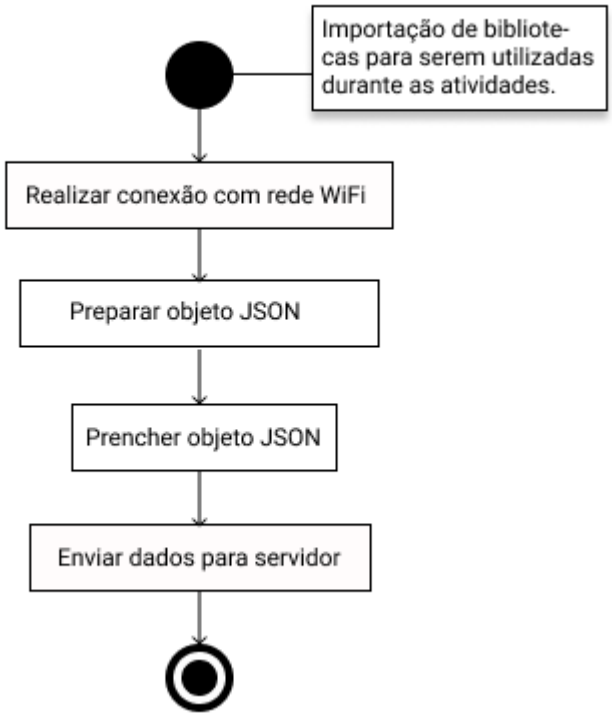
Figura 30 - Gerenciador de Bibliotecas



Fonte: Bauermeister (2017)

Com a finalização da instalação da biblioteca “ArduinoJSON”, é possível dar continuidade ao desenvolvimento do módulo de comunicação. Para exemplificar o funcionamento geral desta etapa, a Figura 31 mostra o diagrama de atividades para explicar o funcionamento do coletor. É importante lembrar que o presente trabalho parte do pressuposto que os dados do indicadores já estão preenchidos, de forma que este não aborda, especificamente, o funcionamento dos sensores.

Figura 31 - Diagrama de atividades do módulo de comunicação



Fonte: O autor (2018)

Com base no diagrama, será exibido o código fonte para a compreensão total do módulo de comunicação. Inicialmente como observado no diagrama, ocorre a importação das bibliotecas necessárias para conectar a uma rede sem fio e montar o objeto JSON respectivamente declaradas nas linhas 2 e 3 do trecho de código exibido na Figura 32. Na mesma imagem também é possível perceber que são declaradas algumas variáveis são elas:

- SSID: variável de tipo char, representa o nome da WiFi que será conectado a placa NodeMCU;
- PASSWORD: variável tipo char, representa a senha da rede WiFi;
- rpiHost: variável tipo char, representa o endereço IP do servidor, para onde os dados serão enviados;
- client: variável tipo WifiClient, proveniente da biblioteca “WiFiClient.h”, cuja declaração é abstraída em função da biblioteca “ESP8266WiFi.h”, é responsável por enviar os dados do sensor;
- jsonBuffer: variável do tipo “StaticJsonBuffer”, proveniente da biblioteca “ArduinoJson.h” que cria a variável “object” de tipo “JsonObject&”, esta

também proveniente da biblioteca “ArduinoJson.h” que será preenchida com os dados do sensor;

Figura 32 - Declaração de bibliotecas e variáveis

```
1   //Programa: Renovar - Modulo de Comunicação - ESP8266 NodeMCU
2   #include <ESP8266WiFi.h> // biblioteca para usar as funções de Wifi do módulo ESP8266
3   #include <ArduinoJson.h>  // biblioteca JSON para sistemas embarcados
4
5
6   // Definições da rede Wifi
7   const char* SSID = "NOME_DA_REDE";
8   const char* PASSWORD = "SENHA_DA_REDE";
9
10
11  // endereço IP local do Servidor Web para onde serão enviados os dados
12  const char* rpiHost = "IP_DO_SERVIDOR";
13
14  WiFiClient client;
15
16  // construindo o objeto JSON que irá armazenar os dados do coletor na função populateJSON()
17  StaticJsonBuffer<300> jsonBuffer;
18  JsonObject& object = jsonBuffer.createObject();
```

Fonte: O autor (2018)

Na Figura 33 é importante destacar principalmente o método “reconnectWiFi”, linha 39, cuja função é realizar a conexão a rede sem fio a partir das variáveis “SSID” e “PASSWORD” declaradas anteriormente e, neste caso, utilizadas como parâmetro .

Figura 33 - Conexão a rede sem fio

```
30  /*
31     função que conecta o NodeMCU na rede Wifi
32     SSID e PASSWORD devem ser indicados nas variáveis
33  */
34  void reconnectWiFi()
35  {
36    if (WiFi.status() == WL_CONNECTED)
37      return;
38
39    WiFi.begin(SSID, PASSWORD);
40
41    while (WiFi.status() != WL_CONNECTED) {
42      delay(100);
43      Serial.print(".");
44    }
45
46    Serial.println();
47    Serial.print("Conectado com sucesso na rede: ");
48    Serial.println(SSID);
49    Serial.print("IP obtido: ");
50    Serial.println(WiFi.localIP());
51  }
```

Fonte: O autor (2018)

A Figura 34 exibe duas funções: “popularJSON” e “realizarPOST”. A primeira é responsável por determinar os parâmetros da variável “object” e atribuí-los com os seus respectivos valores. Já a segunda é responsável por montar a requisição que será enviada ao servidor. A linha 70 mostra o método que realiza a conexão ao servidor, utilizando os parâmetros que represetam IP e porta. Nessa mesma função destaca-se a linha 77 que define: o método HTTP que será utilizado, que método da API será acessado e a versão do protocolo HTTP, e também a linha 83 que preenche o corpo da requisição com valores definidos na função “popularJSON”.

Figura 34 - Montando objeto JSON e realizando Post

```
53  /*
54     função que armazena cada dado do sensor em um objeto JSON
55     utiliza a biblioteca ArduinoJson
56  */
57  void popularJSON()
58  {
59    object["dispositivoId"] = ID_DISPOSITIVO;
60    object["indicadorId"] = ID_INDICADOR;
61    object["medida"] = MEDIDA;
62  }
63
64  /*
65     função que envia os dados do sensor para o servidor em formato JSON
66     faz um POST request ao servidor
67  */
68  void realizarPOST()
69  {
70    if (!client.connect(rpiHost, 8080))    // aqui conectamos ao servidor
71    {
72      Serial.println("Não foi possível conectar ao servidor!n");
73    }
74    else
75    {
76      Serial.println("Conectado ao servidor");
77      client.println("POST /coletas HTTP/1.1");
78      client.println("Host: IP_DO_SERVIDOR");
79      client.println("Content-Type: application/json");
80      client.print("Content-Length: ");
81      client.println(object.measureLength());
82      client.println();
83      object.printTo(client);    // Prenchendo o corpo da requisição, com os dados definidos no método popularJSON()
84    }
85  }
86
```

Fonte: O autor (2018)

Para finalizar o código do coletor ainda há duas funções, que podem ser visualizadas na Figura 35. “Setup” e “loop” são executadas na respectiva ordem e a segunda em repetição infinita. A função “setup” realiza as configurações necessárias para a execução do código, ou seja, após a declaração das variáveis, ela é executada, neste caso, inicializando a conexão Wifi, conforme linha 92. Após a execução de “setup” é iniciado a função “loop”, que engloba a execução da função “popularJSON” e “realizarPOST”, que são sempre repetidas a cada a 10 minutos.

Figura 35 - Funções base

```
86
87  void setup() {
88    pinMode(13, OUTPUT);
89    Serial.begin(115200);
90
91    Serial.println("nIniciando configuração WiFin");
92    initWiFi();
93
94    Serial.println("nConfiguração finalizada, iniciando loopn");
95  }
96
97  void loop() {
98    popularJSON();  // transforma os dados em formato JSON
99    realizarPOST(); // envia os dados ao servidor
100   delay(10000);
101 }
```

Fonte: O autor (2018)

5.3 APLICAÇÃO *FRONT-END*

A aplicação front-end foi a última parte do trabalho a ser desenvolvida, pois era necessário a definição clara da aplicação back-end e do dispositivo de coleta para dar início a esta etapa de implementação. De modo geral a aplicação front-end comunica-se com a aplicação back-end através da API REST Renovar, para buscar os dados necessários, estruturar e apresentá-los de forma geo localizada em uma página web. Esta página foi construída com auxílio de várias tecnologias, ferramentas e padrões de projetos que serão apresentadas no decorrer dessa seção. O código fonte desta etapa de implementação está disponível no apêndice C do presente trabalho e também no repositório do GitHub[10], já as imagens da interface web desenvolvidas encontram-se no apêndice D desse trabalho.

5.3.1 **Ferramentas utilizadas**

A implementação da aplicação front-end só é possível com a instalação dos seguintes softwares:

[10]Código disponível na integra em https://github.com/franciscosft/renovar-webapp

- NodeJS[11]: software utilizado para construir aplicações em JavaScript e gerenciar pacotes utilizados pela linguagem de programação JavaScript, através do NPM;
- VS Code[12]: IDE utilizada para implementação de código.
- Git: utilizado da mesma forma que na aplicação back-end, ou seja, para o versionamento de código;
- Google Chrome: navegador web para visualizar a construção da página;

5.3.2 **Tecnologias utilizadas**

As tecnologias utilizadas para o desenvolvimento da aplicação front-end estão apresentadas a seguir:

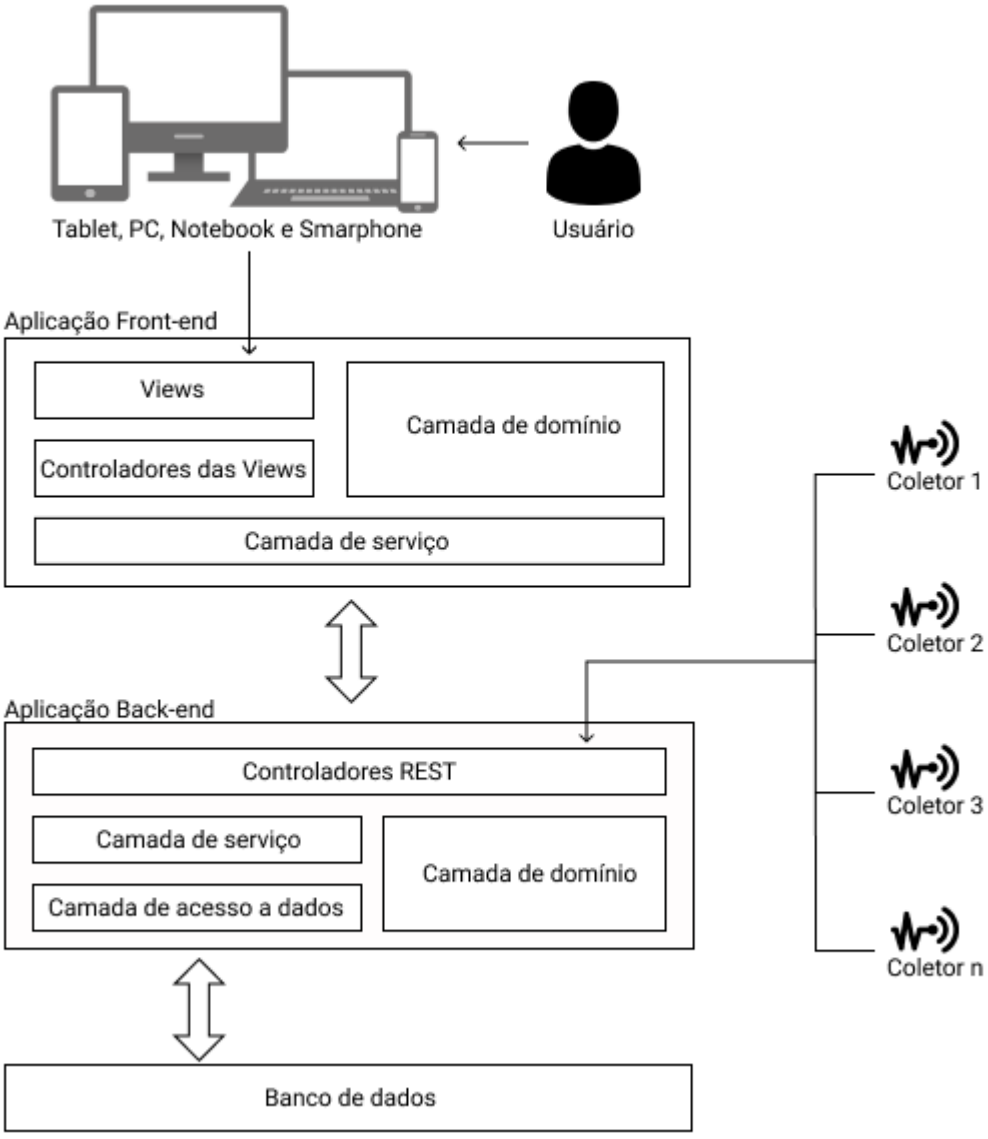
- Angular[13]: framework utilizado para criação de aplicações web;
- Ionic[14]: framework utilizado para a construção de aplicativos híbridos para dispositivos móveis, que por sua vez utiliza o Angular;
- Cordova: API JavaScript que acessa o dispositivo móvel utilizada pelo Ionic;
- Highcharts[15]: API JavaScript utilizada para a criação de gráficos;
- Maps JavaScript API[16]: disponibilizada pelo Google para customizar mapas de acordo com o conteúdo que é desejado ser exibido;
- HTML: linguagem de marcação utilizada para descrever o conteúdo com a sua estrutura
- SASS: linguagem derivada do CSS que permite descrever o estilos que os componentes de uma página web receberão;
- TypeScript: linguagem derivada do JavaScript utilizada pelos frameworks Angular e Ionic;

[11]Mais informações em https://nodejs.org/en/
[12]Mais informações disponíveis em https://code.visualstudio.com/
[13]Mais informações disponíveis em: https://angular.io/
[14]Mais informações disponíveis em https://ionicframework.com/
[15]Mais informações disponíveis em https://www.highcharts.com/
[16]Mais informações disponíveis em: https://developers.google.com/maps/documentation/javascript/tutorial

5.3.3 **Implementação**

Como a aplicação front-end é implementada utilizando Angular encapsulado no Ionic, ela respeita o padrão de projeto MVC, que em tradução direta significa Modelo, Visão e Controle. Este padrão pode ser exemplificado através da Figura 36, que da continuidade a Figura 14, complementando a arquitetura da plataforma Renovar, apresentada de forma simplificada anteriormente.

Figura 36 - Arquitetura completa da Plataforma Renovar



Fonte: O autor (2018)

Na Figura 36 há o detalhamento da aplicação front-end, onde são elucidadas as partes que a compõem. Da mesma forma que a estrutura em camadas utilizada para implementar a aplicação back-end, as partes da aplicação front-end têm funções e responsabilidades específicas, respeitando o padrão de projeto MVC. As funções e responsabilidades são listadas a seguir:

- Modelo: é representado pela camada de domínio, reflete entidades do banco de dados pertinentes a aplicação front-end, trazendo somente atributos necessários para esta etapa do projeto. Também faz parte de Modelo a camada de serviço, ela tem a função de buscar dados na aplicação back-end através da API REST, assim como realizar validações e aplicar a regras de negócio da aplicação;
- Visão: é representada pelas *views*, elas são, simplesmente, responsáveis por exibir a interface web para o usuário através de navegadores de notebooks, tablets, smartphones e computadores;
- Controle: realiza o intermédio entre a visão e o modelo, através dos controladores das views. Eles são responsáveis por compreender as ações do usuário, enviar e/ou buscar dados no modelo.

Com o entendimento do padrão de projeto MVC, é possível dar continuidade as seções subsequentes. As seções a seguir são responsáveis por apresentar, inicialmente, uma visão geral da aplicação front-end, seguida das implementações referentes a cada parte do MVC. Estas foram desenvolvidas para contemplar o funcionamento do MVP da plataforma Renovar.

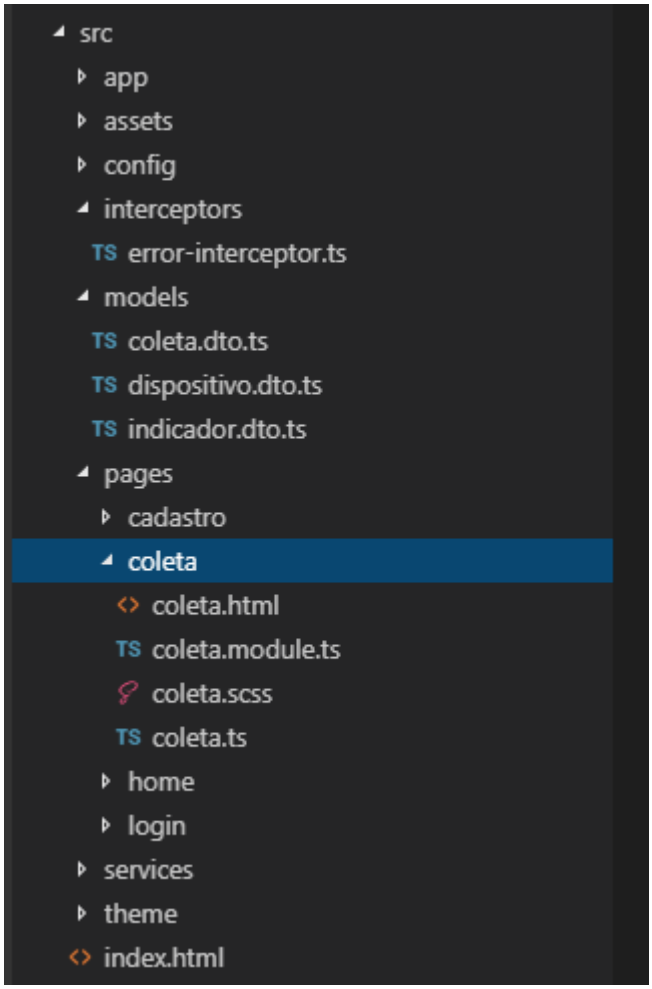
5.3.3.1 Visão Geral

A visão geral tem como objetivo mostrar a estrutura de pastas criadas para a aplicação front-end, identificar quais delas referem-se à cada parte do MVC, além de apresentar as APIs terceiras utilizadas para facilitar na implementação do projeto.

A Figura 37 exibe na sua raiz o diretório “src” expandido, que é responsável por centralizar o código fonte da aplicação, ícones, arquivos de configurações. Nesse diretório que é aplicado o MVC. O modelo é identificado pela pasta “services” e “models”, já a visão e o controle estão juntos em todos o subdiretórios de

“pages”, em que visão é identificada por todos arquivos cuja extensão é “.html” e o controle é identificado pelos arquivos terminados pela extensão “.ts”.

Figura 37 - Visão geral da aplicação front-end



Fonte: O autor (2018)

Foram utilizadas duas APIs terceiras para auxiliar na implementação da plataforma Renovar: Maps JavaScript API e Highcharts. A primeira é um serviço que possibilitou a customização de mapas, que é usado nos arquivos *“index.html”,* “home.ts” e “home.html”. A segunda oferece um serviço para a construção de gráficos, que é utilizado no arquivo “coleta.ts”.

Seguindo o modelo da aplicação back-end, não será apresentado todo código implementado nesta etapa do projeto, a fim de evitar redundância de explicações. É importante lembrar que o código fonte da aplicação front-end está disponível no apêndice C do presente trabalho.

5.3.3.2 Modelo

Esta etapa da aplicação front-end é composta por classes que refletem as entidades do modelo de banco dados e que se encaixam na camada de domínio. Elas são representas por arquivos que seguem o mesmo padrão do trecho de código visualizado na Figura 38. A classe “coleta.dto.ts” é análoga a classe “ColetaRequisicaoDTO” da aplicação back-end, ou seja ambas tem o mesmo objetivo: utilizar somente atributos necessários para o seu respectivo contexto.

Figura 38 - DTO na aplicação Front-end

Fonte: O autor (2018)

Também, nesta etapa, há classes que são responsáveis por utilizar a API REST Renovar, que acessam os recursos da aplicação back-end. Elas são representadas pelos arquivos terminados em “services.ts”, que correspondem a camada de serviço apresentada na Figura 36.

As classes da camada de serviço seguem o modelo de implementação do arquivo “coleta.service.ts”, visualizado conforme trecho de código representado na Figura 39. Neste caso são buscadas todas as coletas dado um intervalo, um identificador de dispositivo e indicador se as variáveis “inicio” e “fim” estiverem preenchidas, caso contrário são buscadas as coletas dado um identificador de dispositivo e indicador.

Figura 39 - Front-end acessando back-end

```
@Injectable()
export class ColetaService {

    constructor(public http: HttpClient) {
    }

    findAllByDispositivo(idDispositivo): Observable<ColetaDTO[]> {···
    }

    findAllByDispositivoIndidicador(idDispositivo, idIndicador, inicio: string, fim: string): Observable<ColetaDTO[]> {
        if (inicio == "" && fim == "" ) {
            console.log("entrou");
            return this.http.get<ColetaDTO[]>(`${API_CONFIG.baseUrl}/coletas/${idDispositivo}/${idIndicador}`);
        } else {
            console.log("sucesso");
            return this.http.get<ColetaDTO[]>(`${API_CONFIG.baseUrl}/coletas/intervalo/
                                              ?idDispositivo=${idDispositivo}&idIndicador=${idIndicador}&dataInicio=${inicio}&dataFim=${fim}`);
        }
    }
}
```

Fonte: O autor (2018)

5.3.3.3 Visão

Como falado anteriormente a visão é identificada pelos arquivos de extensão “.html”, que são construídos com base no framework Ionic. Este possui uma estrutura que permite desenvolver aplicações com interfaces responsivas, ou seja, independente do dispositivo e plataforma utilizada para acessar a aplicação front-end os componentes de uma interface Ionic irão aumentar, diminuir e se organizar automaticamente. Observa-se então que não há necessidade de implementar diferentes tipos de views, para diferentes tipos de dispositivos.

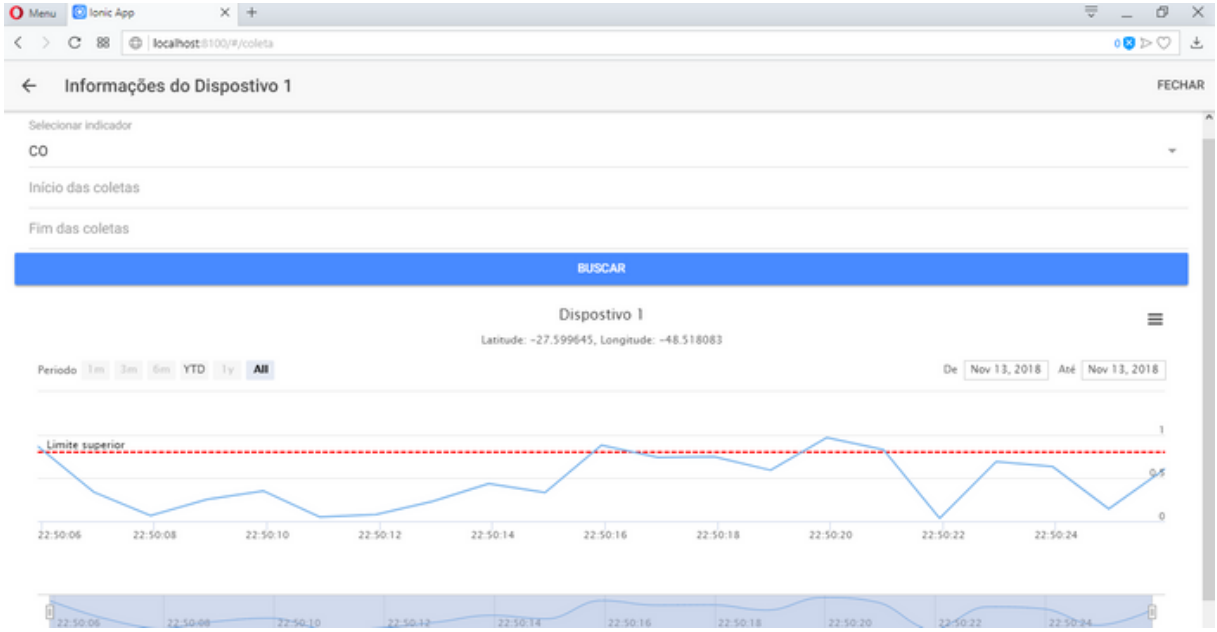
Figura 40 - Coleta HTML

```
<ion-header>
  <ion-navbar>
    <ion-title>Informações do {{dispositivo.nome}}</ion-title>
    <ion-buttons start>
      <button ion-button (click)="dismiss()"> Fechar</button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<script src="https://code.highcharts.com/stock/modules/exporting.js"></script>
<ion-content padding>
  <ion-item>
    <ion-label floating>Selecionar indicador</ion-label>
    <ion-select [(ngModel)]="indicador" interface="alert" submitText="Confirmar" cancelText="Cancelar">
      <ion-option *ngFor="let item of dispositivo.indicadores" [value]="item">{{item.nome}}</ion-option>
    </ion-select>
  </ion-item>
  <ion-item>
    <ion-label>Início das coletas</ion-label>
    <ion-datetime displayFormat="DD/MM/YYYY" [(ngModel)]="event.yestarday"></ion-datetime>
  </ion-item>
  <ion-item>
    <ion-label>Fim das coletas</ion-label>
    <ion-datetime displayFormat="DD/MM/YYYY" [(ngModel)]="event.today"></ion-datetime>
  </ion-item>
  <button ion-button block (click)="buscarColetas(indicador.nome, indicador.id, event.yestarday, event.today)">Buscar</button>
  <ion-item>
    <div id="container"></div>
  </ion-item>
</ion-content>
```

Fonte: O autor (2018)

A Figura 40 mostra o trecho de código presente no arquivo “coleta.html”, responsável por renderizar a página web que exibe as informações do dispositivo de captura com as suas respectivas coletas. Este arquivo HTML utiliza *tags* iniciadas com a palavra “*ion”,* indicando o uso do framework Ionic, que ao ser processado pelo navegador fica conforme a Figura 41 sem ser aplicado a responsividade.

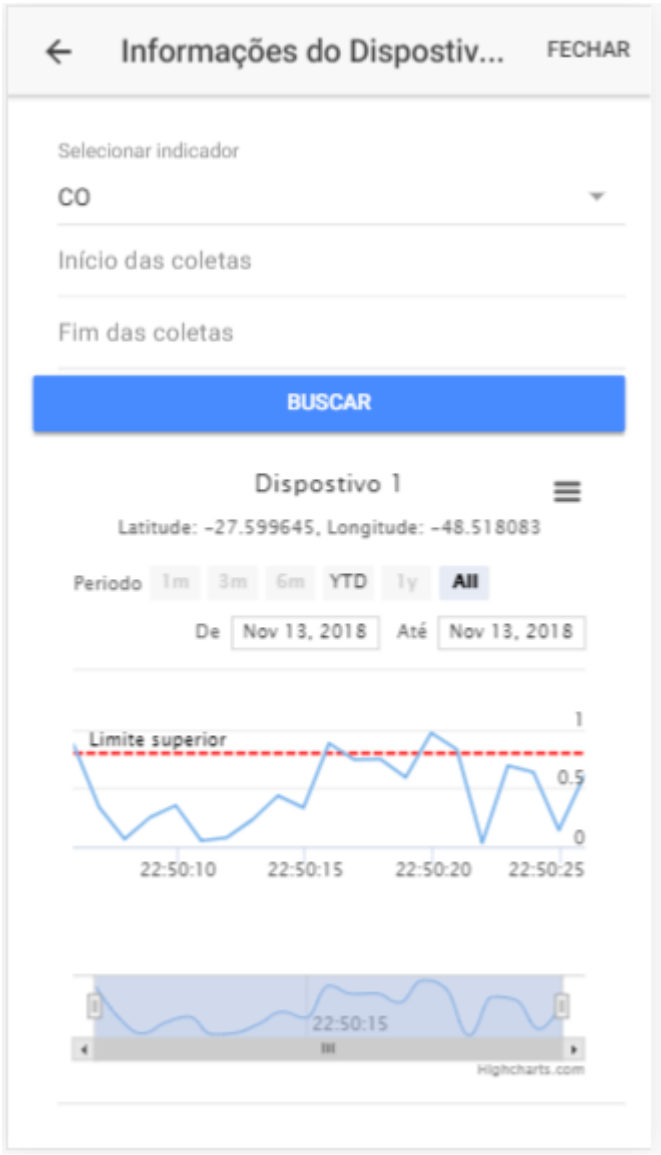
Figura 41 - Interface de Informações do Dispositivo sem responsividade



Fonte: O autor (2018)

Já a Figura 42 exibe o funcionamento da responsividade ao utilizar esta função disponibilizada pelos navegadores. Desta forma é possível perceber na prática que o framework Ionic é uma boa opção para criar aplicações hibridas.

Figura 42 - Interface de Informações do Dispositivo com responsividade



Fonte: O autor (2018)

5.3.3.4 Controle

A etapa de controle da aplicação front-end é exemplificada a partir do arquivo “coleta.ts”, responsável por controlar a sua respectiva visão, a “coleta.html”, abordada na seção anterior. O trecho de código da Figura 43 mostra o método “buscarColetas”, que é executado toda vez que o botão buscar, visualizado na Figura 42, é clicado. A ação deve ser executada após selecionar o indicador que é desejado acompanhar o monitoramento, sendo opcional preenchimento das datas de início e fim que auxiliam a filtrar as coletas.

Figura 43 - Métodos buscar coletas

```
50
51  buscarColetas(nomeIndicador: string, idIndicador: number, inicio: string, fim: string) {
52    this.indicadorService.findById(idIndicador).subscribe(response => {
53      this.indicador = response;
54    });
55    console.log("inicio", inicio);
56    console.log("fim ", fim);
57    this.coletaService.findAllByDispositivoIndidicador(this.dispositivo.id, idIndicador, inicio, fim).subscribe(response => {
58      console.log(response);
59      this.coletas = response;
60
61      var i = 0;
62      var medidas = new Array();
63      this.coletas.forEach(c => {
64        const medida = c.medida;
65        var data = c.data;
66        medidas[i] = [data, medida];
67        i++;
68      });
69
70      this.renderizar(this.indicador, medidas);
71
72    }, error => { });
73  }
74
```

Fonte: O autor (2018)

Com o preenchimento dos campos a visão, informa ao controlador os parâmetros de busca. Este executa o método “findAllByDispositivoIndicador “ visualizado na linha 52 do trecho de código da Figura 43, que realiza uma requisição a aplicação back-end e ao obter a resposta executa o método “renderizar” observado na linha 70 do trecho de código da Figura 43. O método renderizar constrói o gráfico visualizado na Figura 42, a partir dos dados advindos da resposta da aplicação back-end e com auxilio da API Highcharts.

6 **CONSIDERAÇÕES FINAIS**

O MVP da plataforma Renovar foi desenvolvido para o presente TCC no qual fez-se possível colocar em prática diversos conteúdos, conceitos e experiências vistos durante a graduação em Sistemas de Informação, assim como no mercado de trabalho.

Foram utilizadas diversas tecnologias e ferramentas que permitiram tornar a plataforma viável, escalável e replicável. Muitas das tecnologias são open-source e, quando não são seu uso é livre para iniciativas sem fins lucrativos, ou para projetos com viés educacional, como é o caso da Maps JavaScript API.

Também é importante ressaltar que a plataforma atua como uma ponte entre as áreas: IoT, meio ambiente e cidades inteligentes. Uma vez que ela oferece a estrutura básica para disponibilizar dados referentes a qualidade do ar, assim como permite que pessoas apoiem o projeto montando o seu próprio dispositivo de coleta. Dessa maneira, a plataforma atua de forma colaborativa, contribuindo para a construção de uma ciência cidadã, de uma cidade com mais dados e consequentemente mais inteligente.

A partir da implementação da aplicação back-end, front-end e do módulo de comunicação do dispositivo de coleta foi possível atingir os objetivos definidos na Introdução do presente trabalho. Na seção Resultados é feito uma discussão para explicar de que forma cada objetivo foi contemplado.

6.1 RESULTADOS

O primeiro objetivo que foi atingido: o desenvolvimento do módulo para coletar e armazenar dados de sensores que medem a qualidade do ar. Este objetivo é considerado cumprido em função da:

- Implementação do banco de dados;
- Implementação da aplicação back-end que oferece a API REST Renovar;
- Implementação do módulo de comunicação do dispositivo de coleta;

Ainda é importante destacar que a aplicação back-end é generalista, ou seja, ela permite que outras grandezas ou poluentes sejam monitorados. Dessa forma o MVP da plataforma Renovar não fica restrito apenas a dados de qualidade do ar. Isso significa que a aplicação back-end é patível de escala e percebe-se que ela

pode ser utilizada como agrupadora de diversos dados referentes ao meio ambiente.

O segundo objetivo que foi contemplado integralmente: o desenvolvimento de um módulo web para visualizar os relatórios a partir dos dados coletados. Esse objetivo tornou-se possível com a implementação da aplicação front-end, que funciona somente se a aplicação back-end estiver sendo executada. Percebe-se então que a aplicação front-end é dependente da back-end, contudo o contrário não se aplica.

As duas aplicações oferecem um alto valor agregado ao LCQAr, pois a partir de uma visão micro elas possibilitam uma economia de tempo e uma otimização no processo de monitoramento de dados ambientais que o laboratório faz. Uma vez que não há mais a necessidade de utilizar cartão SD nos dispositivos de monitoramento, nem o transporte do cartão para salvar as informações em um computador. A aplicação back-end oferece a API REST Renovar para o dispositivo de coleta enviar os dados, posteriormente serem armazenados no banco de dados. A aplicação front-end oferece a interface web para monitorar os dados e, também realizar o download deles quando necessário.

Também é possível analisar o MVP da plataforma Renovar a partir de uma visão macro. De um lado percebe-se uma grande oportunidade da plataforma ser utilizada pelos cidadãos, a fim de observar dados de qualidade do ar e cobrar medidas as entidades responsáveis e, também contribuindo construindo os seus dispositivos de monitoramento. Por outro lado, o Estado pode utilizar a plataforma, se necessário, para auxiliar na construção de políticas públicas e solucionar problemas referentes a qualidade do ar.

O terceiro e último objetivo que foi concluído: propor uma documentação de um modelo escalável e colaborativo a partir dos módulos previamente desenvolvidos. Dessa forma o presente Trabalho de Conclusão de Curso é considerado o registro formal dessa documentação. O TCC registra todas as etapas de implementação explicando de forma geral cada uma delas, além disso nos apêndices desse trabalho é disponibilizado o código fonte, que pode ser continuado, aperfeiçoado a partir de trabalhos futuros.

6.2 TRABALHOS FUTUROS

Para aplicações que envolvem IoT a segurança é um aspecto importante, em função disso sugere-se um trabalho focado especialmente na segurança entre o dispositivo de coleta e a API REST Renovar. Dessa forma, os dados enviados evitam trafegar sem estarem cifrados e protegidos de uma ponta a outra. Ainda sobre a comunicação entre o dispositivo de coleta e a aplicação back-end, é proposto também uma outra abordagem incluindo um *gateway* entre as pontas, para realizar um possível tratamento e filtros de dados. Dessa forma a arquitetura da plataforma Renovar pode ficar mais flexível e genérica. Além disso, outro fator que apoia a generalização da aplicação é substituir a abstração de banco de dados para armazenamento, uma vez que podem ser utilizadas várias outras abordagens de persistência como NoSQL, armazenamento de arquivos, entre outras.

O desenvolvimento dos trabalhos relacionados permitiu que o autor encontrasse outras plataformas de monitoramento como SmartCitizen e SafeCast. A partir disso outro trabalho que pode ser desenvolvido está relacionado com a integração de dados de diferentes plataformas. Também podem ser construídos trabalhos focados na análise de dados, aplicando algoritmos de aprendizagem de máquina, inteligência artificial entre outros.

Também é importante apontar que um trabalho de experiência com o usuário pode ser realizado futuramente, focado em testes e opiniões a partir da percepção de quem vai utilizar a plataforma. Ainda, sugere-se como trabalho futuro o desenvolvimento de um projeto focado no *design* da aplicação front-end, oferecendo uma nova proposta de layout para interfaces web.

REFERÊNCIAS

AFONSO, Ricardo Alexandre et al. Br-SCMM: Modelo Brasileiro de Maturidade para Cidades Inteligentes. **Simpósio Brasileiro De Sistemas De Informação**, 2013.

ASÍN, Alicia ; CALAHORRA, Manuel . Sensor networks to monitor air pollution in cities. **Libelium**. 2010. Disponível em: <http://www.libelium.com/smart_cities_wsn_air_pollution/>. Acesso em: 3 dez. 2018.

ATZORI, Luigi; MORABITO, Giacomo; IERA, Antonio. The internet of things: A survey. **Computer networks**, 2010.

AZUAGA, Denise. **Danos ambientais causados por veículos leves no Brasil**. 2000. Tese (Engenharia) - UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2000.

BAUERMEISTER, Giovanni. Como usar acelerômetro com ESP8266 NodeMCU. **FILIPEFLOP**. 2017. Disponível em: <https://www.filipeflop.com/blog/acelerometro-com-esp8266-nodemcu/>. Acesso em: 3 dez. 2018.

BNDES. Internet das coisas: estimando impactos na economia . **BNDES O Banco nacional do desenvolvimento**. 2017. Disponível em: <https://www.bndes.gov.br/wps/portal/site/home/conhecimento/noticias/noticia/internet-coisas-iot>. Acesso em: 3 dez. 2018.

BORGIA, Eleonora. The Internet of Things vision: Key features, applications and open issues. **Computer Communications**, 2014.

BOUSKELA, Mauricio et al. Caminho para as smart cities: Da gestão tradicional para a cidade inteligente. **Monografía del BID (Sector de Cambio Climático y Desarrollo Sostenible. División de Viviendas y Desarrollo Urbano)**, 2016.

BRASIL. Ministério do Meio Ambiente. Resolução CONAMA n. 3 28 de junho de 1990. **Diário Oficial da União**22 de agosto de 1990. Disponível em: <http://www2.mma.gov.br/port/conama/legiabre.cfm?codlegi=100>. Acesso em: 3 dez. 2018.

_____. Ministério do Meio Ambiente. Resolução CONAMA n. 306 05 de julho de 2002. **Diário Oficial da União**19 de julho de 2002. Disponível em: <http://www2.mma.gov.br/port/conama/res/res02/res30602.html>. Acesso em: 3 dez. 2018.

CODECADEMY. Back-end Architecture. **Codecademy**. Disponível em: <https://www.codecademy.com/articles/back-end-architecture>. Acesso em: 3 dez. 2018.

_____. What is Rest?: Learn about how to design web services using the REST paradigm. **Codecademy**. Disponível em: <https://www.codecademy.com/articles/what-is-rest>. Acesso em: 3 dez. 2018.

COMANDULLI, Carolina et al. CIÊNCIA CIDADÃ EXTREMA: UMA NOVA ABORDAGEM. **Biodiversidade Brasileira**, p. 34-47, 2016.

CONNECT SMART CITIES. O QUE É O RANKING CONNECTED SMART CITIES?. **Connect Smart Cities**. 2017. Disponível em:<http://www.connectedsmartcities.com.br/o-que-e-o-ranking-connected-smart-cities/>. Acesso em: 4 dez. 2018.

DMITRUK, Hilda Beatriz (Org.). **Cadernos metodológicos**: diretrizes da metodologia científica. 5. ed. Chapecó: Argos, 2001. 123 p.

ENVIROMENT DEFENSE FOUND. Why new sensor technology is critical for tackling air pollution. **EDF - Enviroment Defense Found**. Disponível em: <https://www.edf.org/airqualitymaps>. Acesso em: 3 dez. 2018.

FERREIRA, Cássia Castro Martins; OLIVEIRA, Daiane Evangelista. ESTIMATIVA DA POLUIÇÃO VEICULAR E QUALIDADE DO AR NAS PRINCIPAIS VIAS DO SISTEMA VIÁRIO DA REGIÃO CENTRAL DA CIDADE DE JUIZ DE FORA–MG. **Revista do Departamento de Geografia**, p. 98-114, 2016.

I-BLADES. Air Quality. **I-BLADES**. 2018. Disponível em: <https://i-blades.com/pages/portable-air-quality-monitor-technology>. Acesso em: 3 dez. 2018.

IESE. Cities in Motion. **IESE Business School. University of Navarra**. 2014. Disponível em: <http://www.ieseinsight.com/doc.aspx?id=1582>. Acesso em: 14 jun. 2017.

JENKINS, Edgar W. School science, citizenship and the public understanding of science. **International journal of science education**, p. 703-710, 1999.

JUNCKES, Darlan ; TEIXEIRA, Clarissa Stefani . MODELO BRASILEIRO DE MATURIDADE PARA CIDADES INTELIGENTES: ANÁLISE DOS MUNICÍPIOS DO ESTADO DE SANTA CATARINA. **Revista Eletrônica do Alto Vale do Itajaí**, Dezembro 2016. Disponível em: <http://www.revistas.udesc.br/index.php/reavi/issue/view/498>. Acesso em: 13 out. 2018.

JÚNIOR, Hélio Santiago Ramos; GALIOTTO, Simone. Iniciativas pontuais de cidades inteligentes no meio-oeste catarinense: estudo do caso da cidade de Erval

Velho, a Capital Catarinense da Reciclagem. **Revista Democracia Digital e Governo Eletrônico**, 2013.

KOMNINOS, Nicos. Intelligent cities:: innovation, knowledge systems and digital spaces. **Routledge**, 2013.

LOUREIRO, Antonio A.F. et al. Redes de Sensores Sem Fio. **Simpósio Brasileiro de Redes de Computadores (SBRC)**, p. 179-226, 2003.

METTZER. O melhor editor para trabalhos acadêmicos já feito no mundo. **Mettzer**. Florianópolis, 2016. Disponível em: <http://www.mettzer.com/>. Acesso em: 21 ago. 2016.

OPENDUSTMAP. Citizen-powered air quality mapping. **OpenDustMap**. São Paulo, 2015. Disponível em: <opendustmap.com>. Acesso em: 3 dez. 2018.

PRIBERAM. Sensor. **Dicionário Priberam da Língua Portuguesa**. Disponível em: <https://dicionario.priberam.org/sensor>. Acesso em: 3 dez. 2018.

PUBLIC LAB. DustDuino. **Public Lab**. Disponível em: <https://publiclab.org/wiki/dustduino>. Acesso em: 3 dez. 2018.

RIES, Eric. **The lean startup**: How today's entrepreneurs use continuous innovation to create radically successful businesses. Crown Books, 2011.

SAFECAST. ABOUT SAFECAST. **SAFECAST**. 2018. Disponível em: <https://blog.safecast.org/about/>. Acesso em: 3 dez. 2018.

SALEEMA, Yasir et al. Exploitation of Social IoT for Recommendation Services. **2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)**, Dezembro 2016.

SANTINI, Daniel. Primeiro sensor independente para monitorar poluição é instalado em SP. **Código Urbano**. São Paulo, 2015. Disponível em: <https://www.codigourbano.org/primeiro-sensor-independente-para-monitorar-poluicao-e-instalado-em-sp/>. Acesso em: 3 dez. 2018.

SILVA, Jadson Freire da; FERREIRA, Henrique dos Santos ; SANTOS, Marcelo Olímpio dos . Considerações sobre os estudos em clima urbano. **GEAMA** , Setembro 2015. Disponível em: <http://www.journals.ufrpe.br/index.php/geama/article/view/536>. Acesso em: 13 out. 2018.

SMART CITIZEN. About. **Smart Citizen**. Disponível em: <https://smartcitizen.me/about>. Acesso em: 18 jun. 2017.

_____. **Smart Citizen**. 2018. Disponível em: <https://smartcitizen.me/kits/4721>. Acesso em: 3 dez. 2018.

TACO, Glenda Benita Gonzales. **Desenvolvimento de uma metodologia para identificas espacialmente os níveis de emissão de gases derivados de veículos automotores nas áreas urbanas**. 2006. Dissertação (Engenharia) - UNIVERSIDADE DE BRASÍLIA, 2008. Disponível em: <http://repositorio.unb.br/handle/10482/1996>. Acesso em: 3 dez. 2018.

THOMSEM, Adilson. Qual módulo ESP8266 comprar?. **FILIPEFLOP**. 2016. Disponível em: <https://www.filipeflop.com/blog/qual-modulo-esp8266-comprar/>. Acesso em: 3 dez. 2018.

THOMSEN, Adilson. Como programar o NodeMCU com IDE Arduino. **FILIPEFLOP**. Florianópolis, 2016. Disponível em: <https://www.filipeflop.com/blog/programar-nodemcu-com-ide-arduino/>. Acesso em: 3 dez. 2018.

UNEP. **Year Book 2014 emerging issues update**: Air Pollution: World's Worst Environmental Health Risk. United Nations Environmental Program (UNEP), 2014.

VIENNA UNIVERSITY TECHNOLOGY. The smart city model. **European Smart Cities**. 2015. Disponível em: <http://www.smart-cities.eu/>. Acesso em: 3 dez. 2018.

WORLD HEALTH ORGANIZATION. Air pollution. **World Health Organization**. 2016. Disponível em: <http://apps.who.int/gho/data/node.sdg.3-9-viz-1?lang=en>. Acesso em: 3 dez. 2018.

_____. **Health and the Environment**: Addressing the health impact of air pollution. 2015. Disponível em: <http://apps.who.int/iris/bitstream/handle/10665/253206/A68_ACONF2Rev1-en.pdf?sequence=1&isAllowed=y>. Acesso em: 3 dez. 2018.

ZAMBARDA, Pedro. Internet das Coisas: Entenda o conceito e o que muda com a tecnologia.. **TechTudo**. 2014. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2014/08/internet-das-coisas-entendao-conceito-e-o-que-muda-com-tecnologia.html>. Acesso em: 3 dez. 2018.

APÊNDICE A — Implementação Aplicação Back-end

```
//Arquivo pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>com.renovar</groupId>
	<artifactId>renovar</artifactId>
	<version>0.0.2-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>renovar</name>
	<description>Demo project for Spring Boot</description>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.0.0.RELEASE</version>
		<relativePath /> <!-- lookup parent from repository -->
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>

		<dependency>
			<groupId>com.h2database</groupId>
			<artifactId>h2</artifactId>
			<scope>runtime</scope>
		</dependency>
```

```
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
		</dependency>

		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
		</dependency>

		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger2</artifactId>
			<version>2.7.0</version>
		</dependency>

		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger-ui</artifactId>
			<version>2.7.0</version>
		</dependency>

	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>


</project>

//Classe RenovarApplication

package com.renovar;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.renovar.dao.ColetaDAO;
import com.renovar.dao.DispositivoDAO;
import com.renovar.dao.IndicadorDAO;
import com.renovar.dao.UsuarioDAO;
```

```
import com.renovar.domain.Coleta;
import com.renovar.domain.Coordenada;
import com.renovar.domain.Dispositivo;
import com.renovar.domain.Indicador;
import com.renovar.domain.Usuario;
import com.renovar.domain.enums.Unidade;

@SpringBootApplication
public class RenovarApplication implements CommandLineRunner {
	private final Logger log = LoggerFactory.getLogger(RenovarApplication.class);

	public static void main(String[] args) {
		SpringApplication.run(RenovarApplication.class, args);
	}

	@Override
	public void run(String... args) throws Exception {
	}

}

//Classe DevConfig

package com.renovar.config;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;

import com.renovar.services.DBService;

@Configuration
@Profile("dev")
public class DevConfig {
	private final Logger log = LoggerFactory.getLogger(DevConfig.class);


	@Autowired
	private DBService service;

	@Value("${spring.jpa.hibernate.ddl-auto}")
	private String strategy;

	@Bean
	public boolean  instanciarBandoDeDados() throws InterruptedException {
		// Definindo a estratégia para a criação do banco,  se a chave
spring.jpa.hibernate.ddl-auto != create
		log.info("Estratégia definida: {}", strategy);
		if(!"create".equals(strategy)) {
			return false;
		}
```

```
                service.instanciarBancoDeDados();
                return true;
        }
}

//Classe SweggerConfig

package com.renovar.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import springfox.documentation.swagger2.annotations.EnableSwagger2;

@Configuration
@EnableSwagger2
public class SweggerConfig {

        @Bean
   public Docket api() {
      return new Docket(DocumentationType.SWAGGER_2)
        .select()
        .apis(RequestHandlerSelectors.any())
        .paths(PathSelectors.any())
        .build();
   }

}

//Classe ColetaDAO

package com.renovar.dao;

import java.util.Date;
import java.util.List;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.renovar.domain.Coleta;

@Repository
public interface ColetaDAO extends JpaRepository<Coleta, Integer> {

        @Transactional(readOnly = true)
```

```
	@Query("SELECT obj FROM Coleta obj WHERE obj.dispositivo.id = :id ORDER BY obj.data DESC")
	List<Coleta> buscarColetasDispositivo(@Param("id") Integer id);

	@Transactional(readOnly = true)
	@Query("SELECT obj FROM Coleta obj WHERE obj.dispositivo.id = :id ORDER BY obj.data DESC")
	Page<Coleta> buscarColetasDispositivoPorPagina(@Param("id") Integer id, Pageable pageRequest);

	@Transactional(readOnly = true)
	@Query("SELECT obj FROM Coleta obj WHERE obj.dispositivo.id = :idDispositivo AND obj.indicador.id = :idIndicador ORDER BY obj.data ASC")
	List<Coleta> buscarColetasDispositivoIndicador(@Param("idDispositivo") Integer idDispositivo,
			@Param("idIndicador") Integer idIndicador);

	@Transactional(readOnly = true)
	@Query("SELECT obj FROM Coleta obj WHERE obj.dispositivo.id = :idDispositivo AND obj.indicador.id = :idIndicador ORDER BY obj.data ASC")
	Page<Coleta> buscarColetasPorPaginaDispositivoIndicador(@Param("idDispositivo") Integer idDispositivo,
			@Param("idIndicador") Integer idIndicador, Pageable pageRequest);

	@Transactional(readOnly = true)
	@Query("SELECT obj FROM Coleta obj WHERE obj.dispositivo.id = :idDispositivo AND obj.indicador.id = :idIndicador AND obj.data BETWEEN :dataInicio AND :dataFim ORDER BY obj.data ASC")
	List<Coleta> buscarColetasDispositivoIndicadorData(@Param("idDispositivo") Integer idDispositivo,
			@Param("idIndicador") Integer idIndicador, @Param("dataInicio") Date dataInicio,
			@Param("dataFim") Date dataFim);

}


//Classe DispositivoDAO

package com.renovar.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.renovar.domain.Dispositivo;

@Repository
public interface DispositivoDAO extends JpaRepository<Dispositivo, Integer> {

}

//Classe IndicadorDAO

package com.renovar.dao;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.renovar.domain.Indicador;

@Repository
public interface IndicadorDAO extends JpaRepository<Indicador, Integer> {

}

//Classe UsuarioDAO

package com.renovar.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.renovar.domain.Usuario;

@Repository
public interface UsuarioDAO extends JpaRepository<Usuario, Integer> {

	@Transactional(readOnly=true)
	Usuario findByEmail(String email);

}

//Classe Coleta

package com.renovar.domain;

import java.io.Serializable;
import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.fasterxml.jackson.annotation.JsonIgnore;

@Entity
public class Coleta implements Serializable {

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;
	private Double medida;
	@JsonFormat(pattern = "dd/MM/yyyy HH:mm:ss")
```

```
	private Date data;
	private Double latitude;
	private Double longitude;

	@JsonIgnore // utilizado para esconder os objetos Json
	@ManyToOne
	@JoinColumn(name = "dispositivo_id")
	private Dispositivo dispositivo;

	@ManyToOne
	@JoinColumn(name = "indicador_id")
	private Indicador indicador;

	public Coleta() {
	}

	public Coleta(Integer id, Double medida, Date data, Coordenada coordenada,
Dispositivo dispositivo,
			Indicador indicador) {
		super();
		this.id = id;
		this.medida = medida;
		this.data = data;
		this.latitude = coordenada.getLatitude();
		this.longitude = coordenada.getLongitude();
		this.dispositivo = dispositivo;
		this.indicador = indicador;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public Double getMedida() {
		return medida;
	}

	public void setMedida(Double medida) {
		this.medida = medida;
	}

	public Date getData() {
		return data;
	}

	public void setData(Date data) {
		this.data = data;
	}

	@Override
	public int hashCode() {
```

```
		final int prime = 31;
		int result = 1;
		result = prime * result + ((id == null) ? 0 : id.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Coleta other = (Coleta) obj;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		return true;
	}

	public Dispositivo getDispositivo() {
		return dispositivo;
	}

	public void setDispositivo(Dispositivo sensor) {
		this.dispositivo = sensor;
	}

	@Override
	public String toString() {
		return "Coleta [id=" + id + ", medida=" + medida + ", data=" + data + ",
latitude=" + latitude + ", longitude="
				+ longitude + ", dispositivo=" + dispositivo + ", indicador=" +
indicador + "]";
	}

	public Indicador getIndicador() {
		return indicador;
	}

	public void setIndicador(Indicador indicador) {
		this.indicador = indicador;
	}

	public Double getLatitude() {
		return latitude;
	}

	public void setLatitude(Double latitude) {
		this.latitude = latitude;
	}
```

```
	public Double getLongitude() {
		return longitude;
	}

	public void setLongitude(Double longitude) {
		this.longitude = longitude;
	}

	public Coordenada getCoordenada() {
		return new Coordenada(getLatitude(), getLongitude());
	}

}


//Classe Coordenada

package com.renovar.domain;

import java.io.Serializable;

public class Coordenada implements Serializable {

	private static final long serialVersionUID = 1L;
	private Double latitude;
	private Double longitude;

	public Coordenada() {
	}

	public Coordenada(Double latitude, Double longitude) {
		this.setLatitude(latitude);
		this.setLongitude(longitude);
	}

	public Double getLatitude() {
		return latitude;
	}

	public void setLatitude(Double latitude) {
		this.latitude = latitude;
	}

	public Double getLongitude() {
		return longitude;
	}

	public void setLongitude(Double longitude) {
		this.longitude = longitude;
	}

	@Override
	public String toString() {
		return "Coordenada [latitude=" + latitude + ", longitude=" + longitude + "]";
	}
```

```
}

//Classe Dispositivo

package com.renovar.domain;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;

import com.renovar.dto.DispositivoDTO;

@Entity
public class Dispositivo implements Serializable {

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;
	private String nome;
	private String codigoRastreio;
	private Double latitude;
	private Double longitude;

	// Implementar a deleção em cascade
	@ManyToMany
	@JoinTable(name = "DISPOSITVO_INDICADOR", joinColumns = @JoinColumn(name = "dispositvo_id"), inverseJoinColumns = @JoinColumn(name = "indicador_id"))
	private List<Indicador> indicadores = new ArrayList<>();

	@ManyToOne
	@JoinColumn(name = "usuario_id")
	private Usuario usuario;

	public Dispositivo() {
	}

	public Dispositivo(Integer id, String nome, String codigoRastreio, Coordenada coordenada, Usuario usuario) {
		super();
		this.id = id;
		this.nome = nome;
		this.codigoRastreio = codigoRastreio;
```

```
		this.latitude = coordenada.getLatitude();
		this.longitude = coordenada.getLongitude();
		this.usuario = usuario;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getCodigoRastreio() {
		return codigoRastreio;
	}

	public void setCodigoRastreio(String codigoRastreio) {
		this.codigoRastreio = codigoRastreio;
	}

	@Override
	public String toString() {
		return "Dispositivo [id=" + id + ", nome=" + nome + ", codigoRastreio=" +
codigoRastreio + ", latitude="
				+ latitude + ", longitude=" + longitude + ", indicadores=" +
indicadores + ", usuario=" + usuario + "]";
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((id == null) ? 0 : id.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Dispositivo other = (Dispositivo) obj;
		if (id == null) {
```

```
                        if (other.id != null)
                                return false;
                } else if (!id.equals(other.id))
                        return false;
                return true;
        }

        public List<Indicador> getIndicadores() {
                return indicadores;
        }

        public void setIndicadores(List<Indicador> indicadores) {
                this.indicadores = indicadores;
        }

        public Double getLatitude() {
                return latitude;
        }

        public void setLatitude(Double latitude) {
                this.latitude = latitude;
        }

        public Double getLongitude() {
                return longitude;
        }

        public void setLongitude(Double longitude) {
                this.longitude = longitude;
        }

        public Usuario getUsuario() {
                return usuario;
        }

        public void setUsuario(Usuario usuario) {
                this.usuario = usuario;
        }

}

//Classe Indicador


package com.renovar.domain;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

import com.renovar.domain.enums.Unidade;
```

```
@Entity
public class Indicador implements Serializable {

	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;
	private String nome;
	private Integer unidade;
	private Double limite;
	//TODO Incluir parametro de limite

	public Indicador() {
	}

	public Indicador(Integer id, String nome, Unidade unidade, Double limite) {
		this();
		this.id = id;
		this.nome = nome;
		this.unidade = unidade.getId();
		this.limite = limite;
	}

	public Indicador(Integer id, String nome, Unidade unidade) {
		this();
		this.id = id;
		this.nome = nome;
		this.unidade = unidade.getId();
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getUnidade() {
		return Unidade.toUnidade(unidade).getDescricaoUnidade();
	}

	public void setUnidade(Unidade unidade) {
		this.unidade = unidade.getId();
	}
```

```
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((id == null) ? 0 : id.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Indicador other = (Indicador) obj;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		return true;
	}

	@Override
	public String toString() {
		return "Indicador [id=" + id + ", nome=" + nome + ", unidade=" + unidade + "]";
	}

	public Double getLimite() {
		return limite;
	}

	public void setLimite(Double limite) {
		this.limite = limite;
	}

}

//Classe Usuario

package com.renovar.domain;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Usuario implements Serializable {

	private static final long serialVersionUID = 1L;
```

```
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Integer id;
	private String nome;
	private String sobrenome;
	private String email;
	private String senha;

	public Usuario() {
	}

	public Usuario(Integer id, String nome, String sobrenome, String email, String senha)
{
		this.id = id;
		this.nome = nome;
		this.sobrenome = sobrenome;
		this.email = email;
		this.senha = senha;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((id == null) ? 0 : id.hashCode());
		return result;
	}

	@Override
	public boolean equals(Object obj) {
```

```
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Usuario other = (Usuario) obj;
		if (id == null) {
			if (other.id != null)
				return false;
		} else if (!id.equals(other.id))
			return false;
		return true;
	}

	@Override
	public String toString() {
		return "Usuario [id=" + id + ", nome=" + nome + ", sobrenome=" + sobrenome
+ ", email=" + email + "]";
	}

	public String getSobrenome() {
		return sobrenome;
	}

	public void setSobrenome(String sobrenome) {
		this.sobrenome = sobrenome;
	}

	public String getSenha() {
		return senha;
	}

	public void setSenha(String senha) {
		this.senha = senha;
	}

}

//Classe Unidade

package com.renovar.domain.enums;

public enum Unidade {

						TEMPERATURA(1, "graus celcius"),
						PRESSAO(2, "pascal"),
						CONCENTRACAO(3, "ug/m3");

	private Integer id;
	private String unidade;

	private Unidade(Integer id, String unidade) {
		this.id = id;
		this.unidade = unidade;
```

```
	}

	public Integer getId() {
		return id;
	}

	public String getDescricaoUnidade() {
		return unidade;
	}

	public static Unidade toUnidade(Integer id) {
		if (id == null) {
			return null;
		}

		for (Unidade unidade : Unidade.values()) {
			if (unidade.getId().equals(id)) {
				return unidade;
			}
		}

		throw new IllegalArgumentException("Id inválido: " + id);
	}

}


//Classe ColetaRequisicaoDTO

package com.renovar.dto;

import java.io.Serializable;
import java.util.Date;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.renovar.domain.Coleta;
import com.renovar.domain.Coordenada;
import com.renovar.domain.Indicador;

public class ColetaRequisicaoDTO implements Serializable {

	private static final long serialVersionUID = 1L;
	private Double medida;
	private Integer dispositivoId;
	private Integer indicadorId;

	public ColetaRequisicaoDTO() {
	}

	public ColetaRequisicaoDTO(Integer sensorId, Integer indicadorId, Double medida) {
		this.dispositivoId = sensorId;
		this.indicadorId = indicadorId;
		this.medida = medida;
	}
```

```
	public Double getMedida() {
		return medida;
	}

	public void setMedida(Double medida) {
		this.medida = medida;
	}

	public Integer getDispositivoId() {
		return dispositivoId;
	}

	public void setDispositivoId(Integer dispositivoId) {
		this.dispositivoId = dispositivoId;
	}

	public Integer getIndicadorId() {
		return indicadorId;
	}

	public void setIndicadorId(Integer indicadorId) {
		this.indicadorId = indicadorId;
	}

	@Override
	public String toString() {
		return "ColetaRequisicaoDTO [medida=" + medida + ", dispositivoId=" +
dispositivoId + ", indicadorId=" + indicadorId + "]";
	}

}


//Classe ColetaRespostaDTO

package com.renovar.dto;

import java.io.Serializable;
import java.util.Date;

import com.fasterxml.jackson.annotation.JsonFormat;
import com.renovar.domain.Coleta;
import com.renovar.domain.Coordenada;
import com.renovar.domain.Indicador;

public class ColetaRespostaDTO implements Serializable {

	private static final long serialVersionUID = 1L;
	private Integer id;
	private String dispositivo;
	private String indicadorNome;
	private Double medida;
	private String unidade;
```

```
//	@JsonFormat(pattern = "dd/MM/yyyy HH:mm:ss")
	private Long data;
//	private Coordenada coordenada;

	public ColetaRespostaDTO() {
	}

	// Utilizar para o get
	public ColetaRespostaDTO(Coleta coleta) {
		super();
		id = coleta.getId();
		data = coleta.getData().getTime();
		medida = coleta.getMedida();
		dispositivo = coleta.getDispositivo().getNome();
		Indicador indicador = coleta.getIndicador();
		indicadorNome = indicador.getNome();
		unidade = indicador.getUnidade();
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getDispositivo() {
		return dispositivo;
	}

	public void setSensor(String sensor) {
		this.dispositivo = sensor;
	}

	public String getIndicadorNome() {
		return indicadorNome;
	}

	public void setIndicadorNome(String indicadorNome) {
		this.indicadorNome = indicadorNome;
	}

	public String getUnidade() {
		return unidade;
	}

	public void setUnidade(String unidade) {
		this.unidade = unidade;
	}

	public Long getData() {
		return data;
	}
```

```
	public void setData(Long data) {
		this.data = data;
	}

	public Double getMedida() {
		return medida;
	}

	public void setMedida(Double medida) {
		this.medida = medida;
	}

}

//Classe DispositivoDTO

package com.renovar.dto;

import java.io.Serializable;
import java.util.List;

import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

import com.renovar.domain.Coordenada;
import com.renovar.domain.Dispositivo;
import com.renovar.domain.Indicador;

public class DispositivoDTO implements Serializable {

	private static final long serialVersionUID = 1L;

	private Integer id;
	@NotEmpty(message = "Preencha o nome do sensor para adicionar")
	private String nome;
	private String codigoRastreio;
	private Coordenada coordenada;
	@NotNull(message = "Preencha o identificador do usuário que tem o sensor")
	private Integer usuarioId;
	private String usuario;
	private List<Indicador> indicadores;

	public DispositivoDTO() {
	}

	public DispositivoDTO(Dispositivo dispositivo) {
		super();
		this.id = dispositivo.getId();
		this.nome = dispositivo.getNome();
		this.codigoRastreio = dispositivo.getCodigoRastreio();
		this.coordenada = new Coordenada(dispositivo.getLatitude(),
dispositivo.getLongitude());
		this.usuario = dispositivo.getUsuario().getNome();
		this.usuarioId = dispositivo.getUsuario().getId();
	}
```

```
	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getCodigoRastreio() {
		return codigoRastreio;
	}

	public void setCodigoRastreio(String codigoRastreio) {
		this.codigoRastreio = codigoRastreio;
	}

	public Coordenada getCoordenada() {
		return coordenada;
	}

	public void setCoordenada(Coordenada coordenada) {
		this.coordenada = coordenada;
	}

	public String getUsuario() {
		return usuario;
	}

	public void setUsuario(String usuario) {
		this.usuario = usuario;
	}

	public Integer getUsuarioId() {
		return usuarioId;
	}

	public void setUsuarioId(Integer usuarioId) {
		this.usuarioId = usuarioId;
	}

	@Override
	public String toString() {
		return "DispostivoDTO [id=" + id + ", nome=" + nome + ", codigoRastreio=" +
codigoRastreio + ", coordenada="
				+ coordenada + ", usuarioId=" + usuarioId + ", usuario=" +
usuario + "]";
```

```
	}

	public List<Indicador> getIndicadores() {
		return indicadores;
	}

	public void setIndicadores(List<Indicador> indicadores) {
		this.indicadores = indicadores;
	}

}

//Classe UsuarioDTO


package com.renovar.dto;

import java.io.Serializable;

import javax.validation.constraints.Email;
import javax.validation.constraints.NotEmpty;

import com.renovar.domain.Usuario;

public class UsuarioDTO implements Serializable {

	private static final long serialVersionUID = 1L;

	@NotEmpty(message = "Preenchimento obrigatório")
	private String nome;
	private String sobreNome;

	@Email(message = "E-mail inválido")
	private String email;

	public UsuarioDTO() {
	}

	public UsuarioDTO(Usuario usuario) {
		super();
		this.nome = usuario.getNome();
		this.sobreNome = usuario.getSobrenome();
		this.email = usuario.getEmail();
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}

	public String getSobreNome() {
		return sobreNome;
```

```
	}

	public void setSobreNome(String sobreNome) {
		this.sobreNome = sobreNome;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

}

//Classe ColetaResource

package com.renovar.resources;

import java.net.URI;
import java.util.Date;
import java.util.List;
import java.util.stream.Collectors;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.renovar.RenovarApplication;
import com.renovar.domain.Coleta;
import com.renovar.dto.ColetaRequisicaoDTO;
import com.renovar.dto.ColetaRespostaDTO;
import com.renovar.dto.DispositivoDTO;
import com.renovar.services.ColetaService;
import com.renovar.util.RenovarUtils;

import io.swagger.annotations.ApiOperation;

@RestController
@RequestMapping(value = "/coletas")
public class ColetaResource {
	private final Logger log = LoggerFactory.getLogger(ColetaResource.class);
```

```
	@Autowired
	private ColetaService service;

	@RequestMapping(value = "/{id}", method = RequestMethod.GET)
	public ResponseEntity<Coleta> buscarColeta(@PathVariable Integer id) {
		Coleta coleta = service.buscarColeta(id);
		return ResponseEntity.ok().body(coleta);
	}

	@CrossOrigin
	@RequestMapping(value = "/dispositivo/{idDispositivo}", method = RequestMethod.GET)
	public ResponseEntity<List<ColetaRespostaDTO>> buscarColetasDispositivo(@PathVariable Integer idDispositivo) {
		List<Coleta> coletas = service.buscarColetasDispositivo(idDispositivo);
		List<ColetaRespostaDTO> collect = coletas.stream().map(c -> new ColetaRespostaDTO(c))
				.collect(Collectors.toList());
		return ResponseEntity.ok().body(collect);
	}

	@CrossOrigin
	@RequestMapping(value = "/{idDispositivo}/{idIndicador}", method = RequestMethod.GET)
	public ResponseEntity<List<ColetaRespostaDTO>> buscarColetasDispositivoIndicador(
			@PathVariable Integer idDispositivo,
			@PathVariable Integer idIndicador){
		List<Coleta> coletas = service.buscarColetasDispositivoIndicador(idDispositivo, idIndicador);
		List<ColetaRespostaDTO> collect = coletas.stream().map(c -> new ColetaRespostaDTO(c))
				.collect(Collectors.toList());
		return ResponseEntity.ok().body(collect);
	}

	@CrossOrigin
	@RequestMapping(value = "/intervalo/", method = RequestMethod.GET)
	public ResponseEntity<List<ColetaRespostaDTO>> buscarIndicadoresDatas(
			@RequestParam(value = "idDispositivo", defaultValue = "") Integer idDispositivo,
			@RequestParam(value = "idIndicador", defaultValue = "") Integer idIndicador,
			@RequestParam(value = "dataInicio", defaultValue = "") String inicio,
			@RequestParam(value = "dataFim", defaultValue = "") String fim) {

		if(inicio.isEmpty() || fim.isEmpty()) {
			return ResponseEntity.badRequest().body(null);
		}

		Date dataInicio2 = RenovarUtils.toDataInicio(inicio);
		Date dataFim2 = RenovarUtils.toDataFim(fim);
		log.info("Buscando coletas entre {} e {}", dataInicio2, dataFim2);
```

```
		List<Coleta> coletas = service.buscarColetasDispositivoIndicadorData(idDispositivo, idIndicador, dataInicio2, dataFim2);
		List<ColetaRespostaDTO> collect = coletas.stream().map(c -> new ColetaRespostaDTO(c))
				.collect(Collectors.toList());
		return ResponseEntity.ok().body(collect);
	}

	@RequestMapping(method = RequestMethod.GET)
	public ResponseEntity<Page<ColetaRespostaDTO>> buscarPagina(
			@RequestParam(value = "idDispositivo", defaultValue = "") Integer idDispositivo,
			@RequestParam(value = "page", defaultValue = "0") Integer pagina,
			@RequestParam(value = "linesPerPage", defaultValue = "5") Integer linhasPagina,
			@RequestParam(value = "orderBy", defaultValue = "data") String ordenacao,
			@RequestParam(value = "direction", defaultValue = "DESC") String direcao) {
		Page<Coleta> coleta = service.buscarDispositivoPorPagina(idDispositivo, pagina, linhasPagina, ordenacao,
				direcao);
		Page<ColetaRespostaDTO> dtos = coleta.map(c -> new ColetaRespostaDTO(c));
		return ResponseEntity.ok().body(dtos);
	}

	@RequestMapping(value = "/todas", method = RequestMethod.GET)
	public ResponseEntity<List<ColetaRespostaDTO>> buscarTodas() {
		List<Coleta> coleta = service.buscarTodas();
		List<ColetaRespostaDTO> dtos = coleta.stream().map(c -> new ColetaRespostaDTO(c)).collect(Collectors.toList());
		return ResponseEntity.ok().body(dtos);
	}

	/**
	 * Método responsável por buscar a última Coleta de um dispositivo
	 *
	 * @param idDispositivo
	 * @return Coleta
	 */
	@RequestMapping(value = "/dispositivo/ultima/{idDispositivo}", method = RequestMethod.GET)
	public ResponseEntity<Coleta> buscarUltimaColetaDispositivo(@PathVariable Integer idDispositivo) {
		Coleta coleta = service.buscarUltimaColetaDispositivo(idDispositivo);
		return ResponseEntity.ok().body(coleta);
	}

	/**
	 * Método utilizado para inserir uma coleta
	 *
	 * @param coletaDTO
	 * @return
```

```
         */
        @ApiOperation(value = "Método utilizado para adicionar uma coleta")
        @RequestMapping(method = RequestMethod.POST)
        public ResponseEntity<Void> cadastrarColeta(@RequestBody
ColetaRequisicaoDTO coletaDTO) {
                Coleta coleta = service.toColeta(coletaDTO);
                coleta = service.inserir(coleta);
                URI uri =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(coleta.getI
d()).toUri();
                return ResponseEntity.created(uri).build();
        }

        @RequestMapping(value = "/teste", method = RequestMethod.POST)
        public ResponseEntity<Void> test(@RequestBody ColetaRequisicaoDTO teste) {
                log.info("Parametro passado no teste: {}", teste);
                return ResponseEntity.noContent().build();
        }

}

//Classe DispositivoResource


package com.renovar.resources;

import java.net.URI;
import java.util.List;
import java.util.stream.Collectors;

import javax.validation.Valid;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.renovar.domain.Coleta;
import com.renovar.domain.Dispositivo;
import com.renovar.domain.Indicador;
import com.renovar.dto.ColetaRespostaDTO;
import com.renovar.dto.DispositivoDTO;
import com.renovar.services.DispositivoService;
import com.renovar.services.IndicadorService;

@CrossOrigin
```

```
@RestController
@RequestMapping(value = "/dispositivo")
public class DispositivoResource {
	private static Logger log = LoggerFactory.getLogger(DispositivoResource.class);

	@Autowired
	private DispositivoService service;

	/*
	 * Método utilizado para buscar um determinado dispositivo
	 * @param: id
	 *
	 */
	@RequestMapping(value = "/{id}", method = RequestMethod.GET)
	public ResponseEntity<DispositivoDTO> buscarDispositivo(@PathVariable Integer id) {
		Dispositivo dispositivo = service.buscar(id);
		DispositivoDTO dispositivoDTO = service.toDispositivoDTO(dispositivo);
		return ResponseEntity.ok().body(dispositivoDTO);
	}

	@RequestMapping(value = "/todos", method = RequestMethod.GET)
	public ResponseEntity<List<DispositivoDTO>> buscarDispositivos() {
		log.info("Buscando todos os dispositivos");
		List<Dispositivo> dispositivos = service.buscarTodos();
		List<DispositivoDTO> dispositivosDTO = dispositivos.stream().map(s -> service.toDispositivoDTO(s)).collect(Collectors.toList());
		ResponseEntity<List<DispositivoDTO>> response = ResponseEntity.ok().body(dispositivosDTO);
		return response;
	}

	@RequestMapping(method = RequestMethod.POST)
	public ResponseEntity<Void> cadastrarDispositvo(@Valid @RequestBody DispositivoDTO dispositivoDTO) {
		log.info("DispositivoDTO: {}", dispositivoDTO);
		Dispositivo dispositivo = service.toDispositivo(dispositivoDTO);
//		List<Integer> indicadoresId = dispositivoDTO.getIndicadoresId();
//		List<Indicador> indicadores = indicadorService.toIndicadores(indicadoresId);
//		dispositivo.setIndicadores(indicadores);
		dispositivo = service.inserir(dispositivo);
		// Boa prática de desenvolvendo para retornar a URI do novo objeto criado
		URI uri = ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(dispositivo.getId()).toUri();
		return ResponseEntity.created(uri).build();
	}

	@RequestMapping(value = "/{id}", method = RequestMethod.PUT)
	public ResponseEntity<Void> atualizarDispositivo(@RequestBody Dispositivo dispositivo, @PathVariable Integer id) {
		// Definir quais atributos serão utilizados na atualização
		log.info("Dispositivo: {}", dispositivo);
		dispositivo.setId(id);
		dispositivo = service.atualizar(dispositivo);
```

```
		return ResponseEntity.noContent().build();
	}

	@RequestMapping(value = "/pagina", method = RequestMethod.GET)
	public ResponseEntity<Page<DispositivoDTO>> buscarPagina(
			@RequestParam(value = "page", defaultValue = "0") Integer pagina,
			@RequestParam(value = "linesPerPage", defaultValue = "10") Integer
linhasPagina,
			@RequestParam(value = "orderBy", defaultValue = "id") String
ordenacao,
			@RequestParam(value = "direction", defaultValue = "ASC") String
direcao) {
		Page<Dispositivo> dispositivo = service.encontrarPagina(pagina,
linhasPagina, ordenacao, direcao);
		Page<DispositivoDTO> dtos = dispositivo.map(d -> new DispositivoDTO(d));
		return ResponseEntity.ok().body(dtos);
	}

	@RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
	public ResponseEntity<Void> deletarDispositivo(@PathVariable Integer id) {
		service.deletar(id);
		return ResponseEntity.noContent().build();
	}

}

//Classe IndicadorResource

package com.renovar.resources;

import java.net.URI;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.renovar.domain.Indicador;
import com.renovar.domain.Usuario;
import com.renovar.dto.UsuarioDTO;
import com.renovar.services.IndicadorService;
import com.renovar.services.UsuarioService;

@RestController
@RequestMapping(value = "/indicadores")
public class IndicadorResource {
```

```
	@Autowired
	private IndicadorService service;

	@CrossOrigin
	@RequestMapping(value = "/{id}", method = RequestMethod.GET)
	public ResponseEntity<?> buscarIndicador(@PathVariable Integer id) {
		Indicador indicador = service.buscar(id);
		return ResponseEntity.ok().body(indicador);
	}

	@RequestMapping(method = RequestMethod.GET)
	public ResponseEntity<?> buscarIndicadores() {
		List<Indicador> indicador = service.buscarTodos();
		return ResponseEntity.ok().body(indicador);
	}

	@RequestMapping(method = RequestMethod.POST)
	public ResponseEntity<Void> cadastrarIndicador(@RequestBody Indicador
indicador) {
		indicador = service.inserir(indicador);
		// Boa prática de desenvolvendo para retornar a URI do novo objeto criado
		URI uri =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(indicador.
getId())
				.toUri();
		return ResponseEntity.created(uri).build();
	}

	@RequestMapping(value = "/{id}", method = RequestMethod.PUT)
	public ResponseEntity<?> atualizarIndicador(@RequestBody Indicador indicador,
@PathVariable Integer id) {
		indicador.setId(id);
		indicador = service.atualizar(indicador);
		return ResponseEntity.noContent().build();
	}

}

//Classe UsuarioResource

package com.renovar.resources;

import java.net.URI;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;
```

```
import com.renovar.domain.Usuario;
import com.renovar.dto.UsuarioDTO;
import com.renovar.services.UsuarioService;

@RestController
@RequestMapping(value = "/usuarios")
public class UsuarioResource {

	@Autowired
	private UsuarioService service;

	@RequestMapping(value = "/{id}", method = RequestMethod.GET)
	public ResponseEntity<?> buscarUsuario(@PathVariable Integer id) {
		Usuario usuario = service.buscar(id);
		return ResponseEntity.ok().body(usuario);
	}

	@RequestMapping(method = RequestMethod.GET)
	public ResponseEntity<?> buscarUsuarios() {
		List<Usuario> usuarios = service.buscarTodos();
		return ResponseEntity.ok().body(usuarios);
	}

	@RequestMapping(method = RequestMethod.POST)
	public ResponseEntity<Void> cadastrarUsuario(@RequestBody Usuario usuario) {
		usuario = service.inserir(usuario);
		// Boa prática de desenvolvendo para retornar a URI do novo objeto criado
		URI uri =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(usuario.g
etId())
				.toUri();
		return ResponseEntity.created(uri).build();
	}

	// TODO pesquisar como filtrar por data superior e inferior
	@RequestMapping(value = "/pagina", method = RequestMethod.GET)
	public ResponseEntity<Page<UsuarioDTO>> buscarPagina(
			@RequestParam(value = "page", defaultValue = "0") Integer pagina,
			@RequestParam(value = "linesPerPage", defaultValue = "24") Integer
linhasPagina,
			@RequestParam(value = "orderBy", defaultValue = "nome") String
ordenacao,
			@RequestParam(value = "direction", defaultValue = "ASC") String
direcao) {
		Page<Usuario> usuarios = service.encontrarPagina(pagina, linhasPagina,
ordenacao, direcao);
		Page<UsuarioDTO> dtos = usuarios.map(u -> new UsuarioDTO(u));
		return ResponseEntity.ok().body(dtos);
	}

	@RequestMapping(value = "/{id}", method = RequestMethod.PUT)
	public ResponseEntity<?> atualizarUsuario(@RequestBody Usuario usuario,
@PathVariable Integer id) {
		usuario.setId(id);
```

```
		usuario = service.atualizar(usuario);
		return ResponseEntity.noContent().build();
	}

}

//Classe FieldMessage

package com.renovar.resources.exceptions;

import java.io.Serializable;

public class FieldMessage implements Serializable {

	private static final long serialVersionUID = 1L;

	private String fieldName;
	private String message;

	public FieldMessage() {
	}

	public FieldMessage(String fieldName, String message) {
		super();
		this.fieldName = fieldName;
		this.message = message;
	}

	public String getFieldName() {
		return fieldName;
	}

	public void setFieldName(String fieldName) {
		this.fieldName = fieldName;
	}

	public String getMessage() {
		return message;
	}

	public void setMessage(String message) {
		this.message = message;
	}

}

//Classe ResourceExceptionHandler

package com.renovar.resources.exceptions;

import java.io.Serializable;

public class FieldMessage implements Serializable {

	private static final long serialVersionUID = 1L;
```

```
	private String fieldName;
	private String message;

	public FieldMessage() {
	}

	public FieldMessage(String fieldName, String message) {
		super();
		this.fieldName = fieldName;
		this.message = message;
	}

	public String getFieldName() {
		return fieldName;
	}

	public void setFieldName(String fieldName) {
		this.fieldName = fieldName;
	}

	public String getMessage() {
		return message;
	}

	public void setMessage(String message) {
		this.message = message;
	}

}


//Classe StardadError

package com.renovar.resources.exceptions;

import java.io.Serializable;

import com.fasterxml.jackson.annotation.JsonFormat;

public class StandardError implements Serializable {

	private static final long serialVersionUID = 1L;
	private Integer status;
	private String mensagem;

	@JsonFormat(pattern = "dd/MM/yyyy HH:mm:ss")
	private Long timeStamp;

	public StandardError(Integer status, String mensagem, Long timeStamp) {
		super();
		this.status = status;
		this.mensagem = mensagem;
		this.timeStamp = timeStamp;
	}
```

```
	public Integer getStatus() {
		return status;
	}

	public void setStatus(Integer status) {
		this.status = status;
	}

	public String getMensagem() {
		return mensagem;
	}

	public void setMensagem(String mensagem) {
		this.mensagem = mensagem;
	}

	public Long getTimeStamp() {
		return timeStamp;
	}

	public void setTimeStamp(Long timeStamp) {
		this.timeStamp = timeStamp;
	}

}

//Classe ValidationError

package com.renovar.resources.exceptions;

import java.util.ArrayList;
import java.util.List;

public class ValidationError extends StandardError {

	private static final long serialVersionUID = 1L;

	private List<FieldMessage> errors = new ArrayList<>();

	public ValidationError(Integer status, String mensagem, Long timeStamp) {
		super(status, mensagem, timeStamp);
	}

	public List<FieldMessage> getErrors() {
		return errors;
	}

	public void addError(String fieldName, String message) {
		errors.add(new FieldMessage(fieldName, message));
	}

}

//Classe ColetaService
```

```
package com.renovar.services;

import java.util.Comparator;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.Optional;
import java.util.stream.Collectors;

import javax.crypto.CipherInputStream;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Example;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.data.querydsl.binding.OptionalValueBinding;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.renovar.dao.ColetaDAO;
import com.renovar.domain.Coleta;
import com.renovar.domain.Coordenada;
import com.renovar.domain.Dispositivo;
import com.renovar.domain.Indicador;
import com.renovar.dto.ColetaRequisicaoDTO;
import com.renovar.services.exceptions.ObjectNotFoundException;

@Service
public class ColetaService {
	private final Logger log = LoggerFactory.getLogger(ColetaService.class);

	@Autowired
	private ColetaDAO dao;

	@Autowired
	private DispositivoService dispositivoService;

	@Autowired
	private IndicadorService indicadorService;

	public Coleta buscarColeta(Integer idColeta) {
		Optional<Coleta> findById = dao.findById(idColeta);
		return findById.orElseThrow(() -> new ObjectNotFoundException(
				"Coleta não encontrada: " + idColeta + " , Tipo: " +
Coleta.class.getName()));
	}

	public List<Coleta> buscarColetasDispositivo(Integer idDispositivo) {
		List<Coleta> coletas = dao.buscarColetasDispositivo(idDispositivo);
		return coletas;
	}
```

```
	public List<Coleta> buscarColetasDispositivoIndicador(Integer idDispositivo, Integer idIndicador) {
		List<Coleta> coletas = dao.buscarColetasDispositivoIndicador(idDispositivo, idIndicador);
		return coletas;
	}

	public List<Coleta> buscarColetasDispositivoIndicadorData(Integer idDispositivo, Integer idIndicador,
			Date dataInicio, Date dataFim) {
		log.info("Data: {} e fim: {}", dataInicio, dataFim);
		return dao.buscarColetasDispositivoIndicadorData(idDispositivo, idIndicador, dataInicio, dataFim);
	}

	public Page<Coleta> buscarDispositivoPorPagina(Integer idDispositivo, Integer pagina, Integer linhasPagina, String ordenacao, String direcao) {
		PageRequest pageRequest = PageRequest.of(pagina, linhasPagina, Direction.valueOf(direcao), ordenacao);
		return dao.buscarColetasDispositivoPorPagina(idDispositivo, pageRequest);
	}

	public Coleta buscarUltimaColetaDispositivo(Integer idDispositivo) {
		List<Coleta> buscarColetasDispositivo = buscarColetasDispositivo(idDispositivo);
		return buscarColetasDispositivo.get(0);
	}

	public Page<Coleta> encontrarColetasDeDispositivo(Integer pagina, Integer linhasPagina, String ordenacao, String direcao) {
		PageRequest pageRequest = PageRequest.of(pagina, linhasPagina, Direction.valueOf(direcao), ordenacao);
		return dao.findAll(pageRequest);
	}

	public List<Coleta> buscarTodas() {
		return dao.findAll();
	}

	@Transactional
	public Coleta inserir(Coleta coleta) {
		return dao.save(coleta);
	}


	public Coleta toColeta(ColetaRequisicaoDTO coletaDTO) {
		Date data = new Date();
		double medida = coletaDTO.getMedida();
		Integer dispositivoId = coletaDTO.getDispositivoId();
		Integer indicadorId = coletaDTO.getIndicadorId();
		Dispositivo dispositivo = dispositivoService.buscar(dispositivoId);
		Indicador indicador = indicadorService.buscar(indicadorId);
		Coordenada coordenada = new Coordenada(dispositivo.getLatitude(), dispositivo.getLongitude());
```

```
                Coleta coleta = new Coleta(null, medida, data, coordenada, dispositivo, indicador);
                log.info("Coleta para ser adicionada: {}", coleta);
                return coleta;
        }


}


//Classe DBService

package com.renovar.services;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.renovar.RenovarApplication;
import com.renovar.dao.ColetaDAO;
import com.renovar.dao.DispositivoDAO;
import com.renovar.dao.IndicadorDAO;
import com.renovar.dao.UsuarioDAO;
import com.renovar.domain.Coleta;
import com.renovar.domain.Coordenada;
import com.renovar.domain.Dispositivo;
import com.renovar.domain.Indicador;
import com.renovar.domain.Usuario;
import com.renovar.domain.enums.Unidade;

@Service
public class DBService {
        private final Logger log = LoggerFactory.getLogger(DBService.class);

        @Autowired
        private DispositivoDAO sensorDAO;

        @Autowired
        private IndicadorDAO indicadorDAO;

        @Autowired
        private ColetaDAO coletaDAO;

        @Autowired
        private UsuarioDAO usuarioDAO;

        public void instanciarBancoDeDados() throws InterruptedException {
```

```
		log.info("Criando objetos mock");
		Usuario usuario = new Usuario(null, "Francisco", "Teixeira", "franciscosft@gmail.com", "123");
		usuarioDAO.save(usuario);

		Coordenada coordenada = new Coordenada(-27.599645, -48.518083);
		Coordenada coordenada2 = new Coordenada(-27.6001426, -48.5182837);

		Dispositivo dispositivo = new Dispositivo(null, "Dispostivo 1", "abc", coordenada, usuario);
		Dispositivo dispositivo2 = new Dispositivo(null, "Dispostivo 2", "cdf", coordenada2, usuario);
		usuarioDAO.save(usuario);

		Indicador co = new Indicador(null, "CO", Unidade.CONCENTRACAO, 0.8);
		Indicador temperatura = new Indicador(null, "Termometro", Unidade.TEMPERATURA);
		Indicador pressao = new Indicador(null, "Pressao atmosferica", Unidade.PRESSAO);

		// Settando os indicadores que cada sensor vai ter
		dispositivo.getIndicadores().addAll(Arrays.asList(co, temperatura, pressao));
		dispositivo2.getIndicadores().addAll(Arrays.asList(temperatura, pressao));

		indicadorDAO.saveAll(Arrays.asList(co, temperatura, pressao));
		sensorDAO.saveAll(Arrays.asList(dispositivo, dispositivo2));
		Calendar instance = Calendar.getInstance();
		log.info("Adicionando coletas");
		ArrayList<Coleta> coletas = new ArrayList<>();
		for (int i = 0; i <= 20; i++) {
			coletas.add(new Coleta(null, Math.random(), getData(), coordenada, dispositivo, co));
			coletas.add(new Coleta(null, Math.random(), getData(), coordenada, dispositivo, temperatura));
			coletas.add(new Coleta(null, Math.random(), getData(), coordenada, dispositivo, pressao));
			coletas.add(new Coleta(null, Math.random(), getData(), coordenada, dispositivo2, temperatura));
			Thread.sleep(1000);
		}
		log.info("Coletas adicionadas");

		coletaDAO.saveAll(coletas);
	}

	public Date getData() {
		 GregorianCalendar gc = new GregorianCalendar();

		int year = randBetween(2017, 2018);

		gc.set(gc.YEAR, year);

		int dayOfYear = randBetween(1, gc.getActualMaximum(gc.DAY_OF_YEAR));

		gc.set(gc.DAY_OF_YEAR, dayOfYear);
```

```
            System.out.println(gc.get(gc.YEAR) + "-" + (gc.get(gc.MONTH) + 1) + "-" +
gc.get(gc.DAY_OF_MONTH));

            return gc.getTime();
        }

         public static int randBetween(int start, int end) {
            return start + (int)Math.round(Math.random() * (end - start));
          }

}


//Classe DispositivoService

package com.renovar.services;

import java.util.List;
import java.util.Optional;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.stereotype.Service;

import com.renovar.dao.DispositivoDAO;
import com.renovar.domain.Coleta;
import com.renovar.domain.Dispositivo;
import com.renovar.dto.DispositivoDTO;
import com.renovar.services.exceptions.DataIntegrityException;
import com.renovar.services.exceptions.ObjectNotFoundException;

@Service
public class DispositivoService {
        private final Logger log = LoggerFactory.getLogger(DispositivoService.class);

        @Autowired
        private DispositivoDAO dao;

        @Autowired
        private UsuarioService usuarioService;

        public Dispositivo buscar(Integer id) {
                Optional<Dispositivo> findById = dao.findById(id);
                return findById.orElseThrow(() -> new ObjectNotFoundException(
                                "Sensor não encontrado: " + id + " , Tipo: " +
Dispositivo.class.getName()));
        }

        public List<Dispositivo> buscarTodos() {
```

```
            return dao.findAll();
        }

        public Dispositivo inserir(Dispositivo sensor) {
            return dao.save(sensor);
        }

        public Dispositivo atualizar(Dispositivo sensor) {
            // TODO: definir quais são as informações aptas para atualização
            Dispositivo buscar = buscar(sensor.getId());
            buscar.setNome(sensor.getNome());
            return dao.save(buscar);
        }

        public void deletar(Integer id) {
            buscar(id);
            try {
                dao.deleteById(id);
            } catch (DataIntegrityViolationException e) {
                throw new DataIntegrityException("Não é possivel excluir um sensor
que possua entidade dependentes", e);
            }
        }

        public Dispositivo toDispositivo(DispositivoDTO sensorDTO) {
            log.debug("SensorDTO: {}", sensorDTO);
            return new Dispositivo(sensorDTO.getId(), sensorDTO.getNome(),
sensorDTO.getCodigoRastreio(),
                                sensorDTO.getCoordenada(),
usuarioService.buscar(sensorDTO.getUsuarioId()));
        }

        public Page<Dispositivo> encontrarPagina(Integer pagina, Integer linhasPagina,
String ordenacao, String direcao) {
            PageRequest pageRequest = PageRequest.of(pagina, linhasPagina,
Direction.valueOf(direcao), ordenacao);
            return dao.findAll(pageRequest);
        }

        public DispositivoDTO toDispositivoDTO(Dispositivo dispositivo) {
            DispositivoDTO dispositivoDTO = new DispositivoDTO(dispositivo);
            log.info("Indicadores: {}", dispositivo.getIndicadores());
            dispositivoDTO.setIndicadores(dispositivo.getIndicadores());
            return dispositivoDTO;
        }

}


//Classe IndicadorService

package com.renovar.services;

import java.util.ArrayList;
import java.util.List;
```

```
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.renovar.dao.IndicadorDAO;
import com.renovar.domain.Indicador;
import com.renovar.services.exceptions.ObjectNotFoundException;

@Service
public class IndicadorService {

	@Autowired
	private IndicadorDAO dao;

	public Indicador buscar(Integer id) {
		Optional<Indicador> findById = dao.findById(id);
		return findById.orElseThrow(() -> new ObjectNotFoundException(
				"Indicador não encontrado: " + id + " , Tipo: " +
Indicador.class.getName()));
	}

	public List<Indicador> buscarTodos() {
		return dao.findAll();
	}

	public Indicador inserir(Indicador indicador) {
		indicador.setId(null);
		return dao.save(indicador);
	}

	public List<Indicador> toIndicadores(List<Integer> indicadoresId) {
		List<Indicador> indicadores = new ArrayList<>();
		for (Integer id : indicadoresId) {
			Indicador indicador = buscar(id);
			indicadores.add(indicador);
		}
		return indicadores;
	}

	public Indicador atualizar(Indicador indicador) {
		// TODO Auto-generated method stub
		return null;
	}

}

//Classe UsuarioService

package com.renovar.services;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.stereotype.Service;

import com.renovar.dao.UsuarioDAO;
import com.renovar.domain.Usuario;
import com.renovar.services.exceptions.ObjectNotFoundException;

@Service
public class UsuarioService {

	@Autowired
	private UsuarioDAO dao;

	public Usuario buscar(Integer id) {
		Optional<Usuario> findById = dao.findById(id);
		return findById.orElseThrow(() -> new ObjectNotFoundException(
				"Usuario não encontrado: " + id + " , Tipo: " + Usuario.class.getName()));
	}

	public List<Usuario> buscarTodos() {
		return dao.findAll();
	}

	public Usuario inserir(Usuario usuario) {
		usuario.setId(null);
		return dao.save(usuario);
	}

	public Usuario atualizar(Usuario usuario) {
		// TODO: definir quais são as informações aptas para atualização
		Usuario usuarioEncontrado = buscar(usuario.getId());
		usuarioEncontrado.setNome(usuario.getNome());
		usuarioEncontrado.setEmail(usuario.getEmail());
		return dao.save(usuarioEncontrado);
	}

	public Page<Usuario> encontrarPagina(Integer pagina, Integer linhasPagina, String ordenacao, String direcao) {
		PageRequest pageRequest = PageRequest.of(pagina, linhasPagina, Direction.valueOf(direcao), ordenacao);
		return dao.findAll(pageRequest);
	}

}

//Classe ObjectNotFoundException

package com.renovar.services.exceptions;

public class ObjectNotFoundException extends RuntimeException {

	private static final long serialVersionUID = 1L;
```

```
	public ObjectNotFoundException(String mensagem) {
		super(mensagem);
	}

	public ObjectNotFoundException(String mensagem, Throwable cause) {
		super(mensagem, cause);
	}
}

//Classe DataIntegrityException

package com.renovar.services.exceptions;

public class DataIntegrityException extends RuntimeException {

	private static final long serialVersionUID = 1L;

	public DataIntegrityException(String mensagem) {
		super(mensagem);
	}

	public DataIntegrityException(String mensagem, Throwable cause) {
		super(mensagem, cause);
	}
}

//Classe RenovarUtils

package com.renovar.util;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.renovar.services.ColetaService;

public class RenovarUtils {

	private final static Logger log = LoggerFactory.getLogger(RenovarUtils.class);


	public static Date dataFormatada(Date date) {
		SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
		try {
			return sdf.parse(date + "");
		} catch (ParseException e) {
			return date;
		}
	}
```

```
	public static Date toDataInicio(String inicio) {
		try {
			String inicioDia = inicio + " 00:00:00";
			SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss", Locale.ENGLISH);
			return formatter.parse(inicioDia);
		} catch (ParseException e) {
			log.error(e.getMessage(), e);
			return new Date();
		}
	}

	public static Date toDataFim(String fim) {
		try {
			String fimDia = fim + " 23:59:59";
			SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss", Locale.ENGLISH);
			return formatter.parse(fimDia);
		} catch (ParseException e) {
			log.error(e.getMessage(), e);
			return new Date();
		}
	}
}

//Classe URL

package com.renovar.util;

import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;

public class URL {


	public static String decodeParam(String s) {
		try {
			return URLDecoder.decode(s, "UTF-8");
		} catch (UnsupportedEncodingException e) {
			return "";
		}
	}

}

//Arquivo application-properties

spring.datasource.url=jdbc:mysql://localhost:3306/bd_renovar_dev
spring.datasource.username=root
spring.datasource.password=

spring.jpa.hibernate.ddl-auto=none
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

logging.level.org.springframework.web=INFO

APÊNDICE B — Implementação do Módulo de Comunicação

```
//Programa: Renovar - Modulo de Comunicação - ESP8266 NodeMCU
#include <ESP8266WiFi.h> // biblioteca para usar as funções de Wifi do módulo ESP8266
#include <ArduinoJson.h>  // biblioteca JSON para sistemas embarcados

// Definições da rede Wifi
const char* SSID = "NOME_DA_REDE";
const char* PASSWORD = "SENHA_DA_REDE";

// endereço IP local do Servidor Web para onde serão enviados os dados
const char* rpiHost = "IP_DO_SERVIDOR";

WiFiClient client;

// construindo o objeto JSON que irá armazenar os dados do coletor na função populateJSON()
StaticJsonBuffer<300> jsonBuffer;
JsonObject& object = jsonBuffer.createObject();

void initWiFi()
{
        delay(10);
        Serial.print("Conectando-se na rede: ");
        Serial.println(SSID);
        Serial.println("Aguarde");

        reconnectWiFi();
}

/*
  função que conecta o NodeMCU na rede Wifi
  SSID e PASSWORD devem ser indicados nas variáveis
*/
void reconnectWiFi() {
        if (WiFi.status() == WL_CONNECTED)
          return;

        WiFi.begin(SSID, PASSWORD);

        while (WiFi.status() != WL_CONNECTED) {
          delay(100);
          Serial.print(".");
 }

Serial.println();
Serial.print("Conectado com sucesso na rede: ");
Serial.println(SSID);
Serial.print("IP obtido: ");
Serial.println(WiFi.localIP());
}

/*
        função que armazena cada dado do sensor em um objeto JSON
```

```
        utiliza a biblioteca ArduinoJson
*/
void popularJSON() {
        object["dispositivoId"] = ID_DISPOSITIVO;
        object["indicadorId"] = ID_INDICADOR;
        object["medida"] = MEDIDA;
}

/*
        função que envia os dados do sensor para o servidor em formato JSON
        faz um POST request ao servidor
*/
void realizarPOST()
      {
        if (!client.connect(rpiHost, 8080))    // aqui conectamos ao servidor
        {
          Serial.println("Não foi possível conectar ao servidor!n");
        }
        else
        {
          Serial.println("Conectado ao servidor");
          client.println("POST /coletas HTTP/1.1");
          client.println("Host: IP_DO_SERVIDOR");
          client.println("Content-Type: application/json");
          client.print("Content-Length: ");
          client.println(object.measureLength());
          client.println();
            object.printTo(client);      // Prenchendo o corpo da requisição, com os dados
definidos no método popularJSON()
        }
      }

void setup() {
        pinMode(13, OUTPUT);
        Serial.begin(115200);

        Serial.println("nIniciando configuração WiFin");
        initWiFi();

        Serial.println("nConfiguração finalizada, iniciando loopn");
}

void loop() {
        popularJSON();  // transforma os dados em formato JSON
        realizarPOST(); // envia os dados ao servidor
        delay(10000);
}
```

APÊNDICE C — Implementação Aplicação Front-end

```
//Arquivo app.module.ts

import { BrowserModule } from '@angular/platform-browser';
import {HttpClientModule } from '@angular/common/http';
import { ErrorHandler, NgModule } from '@angular/core';
import { IonicApp, IonicErrorHandler, IonicModule } from 'ionic-angular';

import { MyApp } from './app.component';

import { StatusBar } from '@ionic-native/status-bar';
import { SplashScreen } from '@ionic-native/splash-screen';
import { DispositivoService } from '../services/domain/dispositivo.service';
import { ErrorInterceptorProvider } from '../interceptors/error-interceptor';
import { ColetaService } from '../services/domain/coleta.service';
import { IndicadorService } from '../services/domain/indicador.service';

@NgModule({
  declarations: [
    MyApp,
  ],
  imports: [
    BrowserModule,
    HttpClientModule,
    IonicModule.forRoot(MyApp, {
      monthNames: ['janeiro', 'fevereiro', 'março', 'abril', 'maio', 'junho', 'agosto', 'setembro',
'outubro', 'novembro', 'dezembro'],
      monthShortNames: ['jan', 'fev', 'mar', 'abr', 'mai', 'jun', 'ago', 'set', 'out', 'nov', 'dez'],
      dayNames: ['domingo', 'segunda-feira', 'terça-feira', 'quarta-feira', 'quinta-feira', 'sexta-
feira', 'sabado'],
      dayShortNames: ['dom', 'seg', 'ter', 'qua', 'qui', 'sex', 'sab'],
    })
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
  ],
  providers: [
    StatusBar,
    SplashScreen,
    {provide: ErrorHandler, useClass: IonicErrorHandler},
    DispositivoService,
    ColetaService,
    IndicadorService,
    ErrorInterceptorProvider,
  ]
})
export class AppModule {}

//Arquivo api.config.ts

export const API_CONFIG = {
    baseUrl: "http://localhost:8080"
```

```
}

//Arquivo error-interceptor.ts

import { Injectable } from '@angular/core';
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest, HTTP_INTERCEPTORS }
from '@angular/common/http';
import { Observable } from 'rxjs/Rx'; // IMPORTANTE: IMPORT ATUALIZADO
@Injectable()
export class ErrorInterceptor implements HttpInterceptor {

    intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
        console.log("Passou no interceptor");
        return next.handle(req)
        .catch((error, caught) => {

            let errorObj = error;
            if (errorObj.error) {
                errorObj = errorObj.error;
            }
            if (!errorObj.status) {
                errorObj = JSON.parse(errorObj);
            }
            console.log("Erro detectado pelo interceptor:");
            console.log(errorObj);
            return Observable.throw(errorObj);
        }) as any;
    }
}

export const ErrorInterceptorProvider = {
    provide: HTTP_INTERCEPTORS,
    useClass: ErrorInterceptor,
    multi: true,
};

//Arquivo coleta.dto.ts

import { DateTime } from "ionic-angular";

export interface ColetaDTO {
    id : string;
    dispositivo: string;
            indicadorNome : string ;
            medida: number;
    unidade: string ;
    data: number;
}

//Arquivo dispositivo.dto.ts

import { IndicadorDTO } from "./indicador.dto";

export interface DispositivoDTO {
     id : string;
```

```
    nome : string;
    codigoRastreio: string;
    coordenada: {
        latitude : string;
        longitude: string;
    }
    indicadores: IndicadorDTO[];
}

//Arquivo indicador.dto.ts

export interface IndicadorDTO {
    id : string;
    nome : string;
    unidade: string;
    limite: number;
}

//Arquivo coleta.html

<ion-header>
  <ion-navbar>
    <ion-title>Informações do {{dispositivo.nome}}</ion-title>
    <ion-buttons start>
      <button ion-button (click)="dismiss()"> Fechar</button>
    </ion-buttons>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-item>
    <ion-label floating>Selecionar indicador</ion-label>
    <ion-select [(ngModel)]="indicador" interface="alert" submitText="Confirmar"
cancelText="Cancelar">
      <ion-option *ngFor="let item of dispositivo.indicadores"
[value]="item">{{item.nome}}</ion-option>
    </ion-select>
  </ion-item>
  <ion-item>
    <ion-label>Início das coletas</ion-label>
    <ion-datetime displayFormat="DD/MM/YYYY" [(ngModel)]="event.yestarday"></ion-
datetime>
  </ion-item>
  <ion-item>
    <ion-label>Fim das coletas</ion-label>
    <ion-datetime displayFormat="DD/MM/YYYY" [(ngModel)]="event.today"></ion-datetime>
  </ion-item>
  <button ion-button block (click)="buscarColetas(indicador.nome, indicador.id,
event.yestarday, event.today)">Buscar</button>
  <ion-item>
    <div id="container"></div>
  </ion-item>
</ion-content>

//Arquivo coleta.module.ts
```

```
import { NgModule } from '@angular/core';
import { IonicPageModule } from 'ionic-angular';
import { ColetaPage } from './coleta';

@NgModule({
  declarations: [
    ColetaPage,
  ],
  imports: [
    IonicPageModule.forChild(ColetaPage),
  ],
})
export class ColetaPageModule {}

//Arquivo coleta.scss

page-coleta {

    #container {
        width:100%;
        height:400px;
        display: block;
    }

}

//Arquivo coleta.ts
import { Component } from '@angular/core';
import { DispositivoDTO } from "../../models/dispositivo.dto";
import { IonicPage, NavController, NavParams, ViewController, DateTime } from 'ionic-
angular';
import { ColetaService } from '../../services/domain/coleta.service';
import { IndicadorService } from '../../services/domain/indicador.service';
import * as HighCharts from 'highcharts';
import * as HighStock from 'highcharts/highstock';
// import * as Datas from 'highcharts/modules/export-data.js';
import { ColetaDTO } from '../../models/coleta.dto';
import { IndicadorDTO } from '../../models/indicador.dto';

// declare var require: any;
// var hcharts = require('highcharts');
// require('highcharts/modules/exporting')(hcharts);
// require('highcharts/modules/export-data')(hcharts);

declare var require: any;
var hcharts = require('highcharts/highstock');
require('highcharts/modules/exporting')(hcharts);
require('highcharts/modules/export-data')(hcharts);

@IonicPage()
@Component({
  selector: 'page-coleta',
  templateUrl: 'coleta.html',
})
```

```
export class ColetaPage {

  dispositivo: DispositivoDTO;
  coletas: ColetaDTO[];
  indicador: IndicadorDTO;

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    public viewCtrl: ViewController,
    public coletaService: ColetaService,
    public indicadorService: IndicadorService) {
    this.dispositivo = navParams.data;
  }

  public event = {
    yestarday: '',
    today: ''
  }

  ionViewDidLoad() {

  }

  buscarColetas(nomeIndicador: string, idIndicador: number, inicio: string, fim: string) {
    this.indicadorService.findById(idIndicador).subscribe(response => {
      this.indicador = response;
    });
    console.log("inicio", inicio);
    console.log("fim ", fim);
    this.coletaService.findAllByDispositivoIndidicador(this.dispositivo.id,    idIndicador,    inicio,
fim).subscribe(response => {
      console.log(response);
      this.coletas = response;

      var i = 0;
      var medidas = new Array();
      this.coletas.forEach(c => {
        const medida = c.medida;
        var data = c.data;
        medidas[i] = [data, medida];
        i++;
      });

      this.renderizar(this.indicador, medidas);

    }, error => { });
  }



  renderizar(indicador: IndicadorDTO, medidas: any[]): any {
    console.log("Limite: ", indicador.limite);

    HighStock.setOptions({
```

```
  // HighCharts.setOptions({
    lang: {
      loading: 'Aguarde...',
      months: ['Janeiro', 'Fevereiro', 'Março', 'Abril', 'Maio', 'Junho', 'Julho', 'Agosto',
'Setembro', 'Outubro', 'Novembro', 'Dezembro'],
      weekdays: ['Domingo', 'Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta', 'Sábado'],
      shortMonths: ['Jan', 'Feb', 'Mar', 'Abr', 'Maio', 'Jun', 'Jul', 'Ago', 'Set', 'Out', 'Nov', 'Dez'],
      exportButtonTitle: "Exportar",
      printButtonTitle: "Imprimir",
      rangeSelectorFrom: "De",
      rangeSelectorTo: "Até",
      rangeSelectorZoom: "Periodo",
      downloadPNG: 'Download imagem PNG',
      downloadJPEG: 'Download imagem JPEG',
      downloadPDF: 'Download documento PDF',
      downloadSVG: 'Download imagem SVG'
    }
  }
  );

  HighStock.stockChart('container', {
  // HighCharts.chart('container', {

    exporting: {
      chartOptions: { // specific options for the exported image
        plotOptions: {
          series: {
            dataLabels: {
              enabled: false
            }
          }
        }
      },
      buttons: {
        contextButton: {
          menuItems: ['downloadPDF', 'downloadPNG', 'downloadCSV', 'downloadXLS']
        }
      },
      csv: {
        dateFormat: '%Y-%m-%d %H:%M:%S',
        decimalPoint: '.'
      },
      fallbackToExportServer: false
    },

    title: {
      text: this.dispositivo.nome
    },

    subtitle: {
      text: `Latitude: ${this.dispositivo.coordenada.latitude}, Longitude:
${this.dispositivo.coordenada.longitude}`
    },

    navigator: {
```

```
      margin: 60
    },

    yAxis: {
      plotLines: [{
        value: indicador.limite,
        color: 'red',
        dashStyle: 'shortdash',
        width: 2,
        label: {
          text: 'Limite superior'
        }
      },
      ],
    },

    xAxis: {
      type: 'datetime',
      labels: {
        format: '{value: %H:%M:%S}'
      }
    },

    series: [{
      name: indicador.nome,
      data: medidas,
      pointStart: Date.UTC(2010, 0, 1),
      tooltip: {
        valueDecimals: 2,
      },
    }],

  });

 }


 dismiss() {
  this.viewCtrl.dismiss(this.dispositivo);
 }

}

//Arquivo home.html

<script src="https://code.highcharts.com/stock/modules/exporting.js"></script>
<ion-header>
  <ion-navbar>
    <button ion-button menuToggle>
      <ion-icon name="menu"></ion-icon>
    </button>

    <ion-title>Renovar</ion-title>
  </ion-navbar>
</ion-header>
```

```
<ion-content padding>
  <div id="floating-panel">
  </div>
  <div #map id="map"></div>

</ion-content>

//Arquivo home.module.ts
import { IonicPageModule } from 'ionic-angular/module';
import { NgModule } from '@angular/core';
import { HomePage } from './home';
@NgModule({
    declarations: [HomePage],
    imports: [IonicPageModule.forChild(HomePage)]
})
export class HomeModule {
}

//Arquivo home.scss

page-home {
    h3 {
        text-align: center;
    }

    img {
        width: 400px;
        align-self: center;
        margin: auto;
        display: block;
        padding: 10px;
    }

    .acoes {
        top: 10px;
        min-width: 300px;
        max-width: 600px;
        margin: auto;
        padding: 10px;
    }

    #map {
        height: 100%;
      }

     #floating-panel {
        position: absolute;
        top: 10px;
        right: 0;
        z-index: 5;
        background-color: #fff;
        padding: 5px;
        text-align: center;
        font-family: 'Roboto','sans-serif';
```

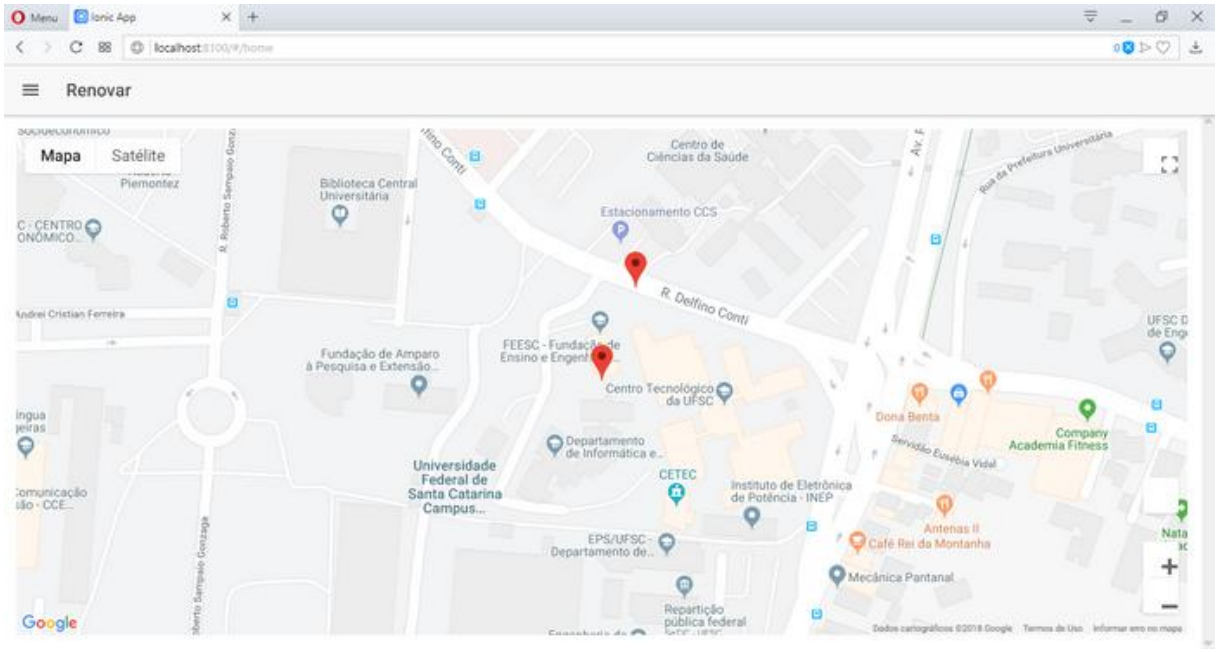
```
      line-height: 30px;
    }
}

//Arquivo home.ts

import { Component, ViewChild, ElementRef } from '@angular/core';
import { NavController, NavParams, ModalController, IonicPage } from 'ionic-angular';
import { DispositivoDTO } from "../../models/dispositivo.dto";
import { DispositivoService } from '../../services/domain/dispositivo.service';


declare var google;

@IonicPage()
@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {

  @ViewChild('map') mapElement: ElementRef;
  map: any;

  constructor(
    public navCtrl: NavController,
    public navParams: NavParams,
    public dispositivoService: DispositivoService,
    public modCtrl: ModalController) {
  }

  ionViewDidLoad() {
    this.dispositivoService.findAll().subscribe(response => {
      this.initMap(response);
    },
    error => {});
  }

  initMap(dispositivos: DispositivoDTO[]) {
    console.log(dispositivos);

    this.map = new google.maps.Map(this.mapElement.nativeElement, {
      zoom: 18,
      center: {lat: -27.6001426, lng: -48.5182837 }
    });

    dispositivos.forEach(dispositivo => {
      console.log(dispositivo.nome);
      var marker = new google.maps.Marker({
        position: {lat: dispositivo.coordenada.latitude, lng: dispositivo.coordenada.longitude },
        map: this.map,
        title: dispositivo.nome
      });

      let that = this;
```

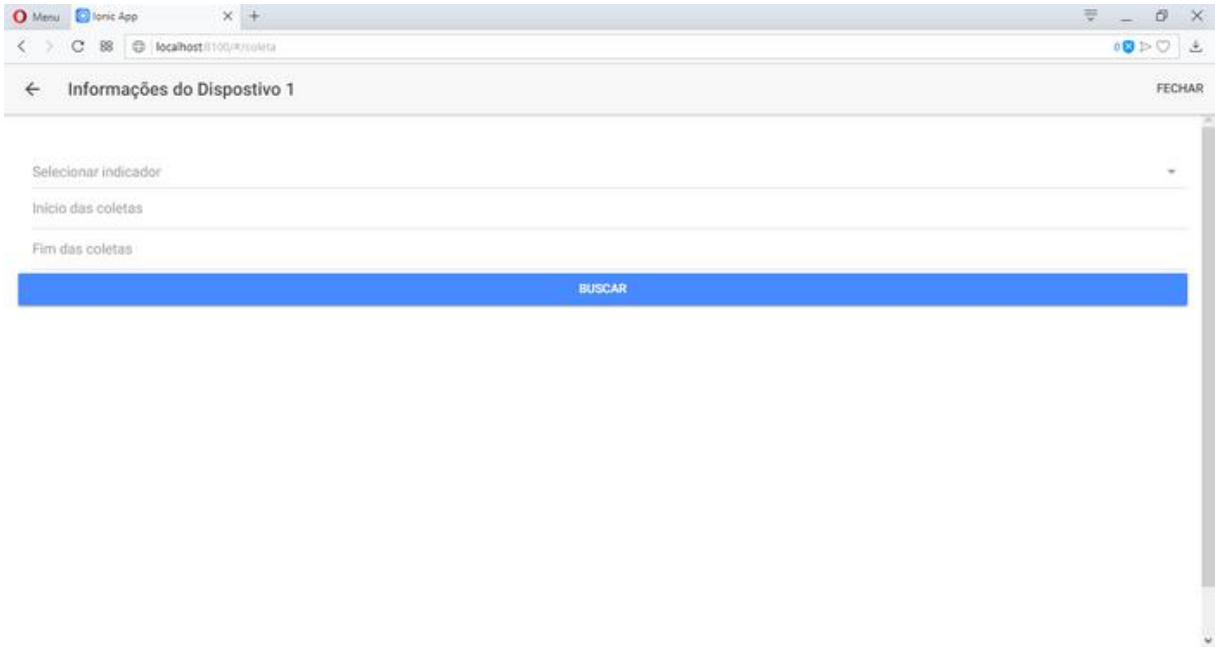
```
      marker.addListener('click', function() {
        // let coletaModal = that.modCtrl.create('ColetaPage', dispositivo);
        // coletaModal.present();
        that.navCtrl.push('ColetaPage', dispositivo)
      });
    });
  }

  login() {
    // let profileModal = this.modCtrl.create('LoginPage');
    // profileModal.present();
    this.navCtrl.push('LoginPage')
  }

  registrar() {
    this.navCtrl.push('CadastroPage')
  }

}

//Arquivo coleta.service.ts

import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { API_CONFIG } from "../../config/api.config";
import { ColetaDTO } from "../../models/coleta.dto";
import { Observable } from "rxjs/Rx";

@Injectable()
export class ColetaService {

    constructor(public http: HttpClient) {
    }

    findAllByDispositivo(idDispositivo): Observable<ColetaDTO[]> {
        return
this.http.get<ColetaDTO[]>(`${API_CONFIG.baseUrl}/coletas/dispositivo/${idDispositivo}`);
    }

    findAllByDispositivoIndidicador(idDispositivo, idIndicador, inicio: string, fim: string):
Observable<ColetaDTO[]> {
        console.log(inicio);
        console.log(inicio.length)
        console.log(fim);
        console.log(fim.length);
        if (inicio == "" && fim == "" ) {
            console.log("entrou");
            return
this.http.get<ColetaDTO[]>(`${API_CONFIG.baseUrl}/coletas/${idDispositivo}/${idIndicador}`
);
        } else {
            console.log("sucesso");
            return
this.http.get<ColetaDTO[]>(`${API_CONFIG.baseUrl}/coletas/intervalo/?idDispositivo=${idDi
spositivo}&idIndicador=${idIndicador}&dataInicio=${inicio}&dataFim=${fim}`);
```

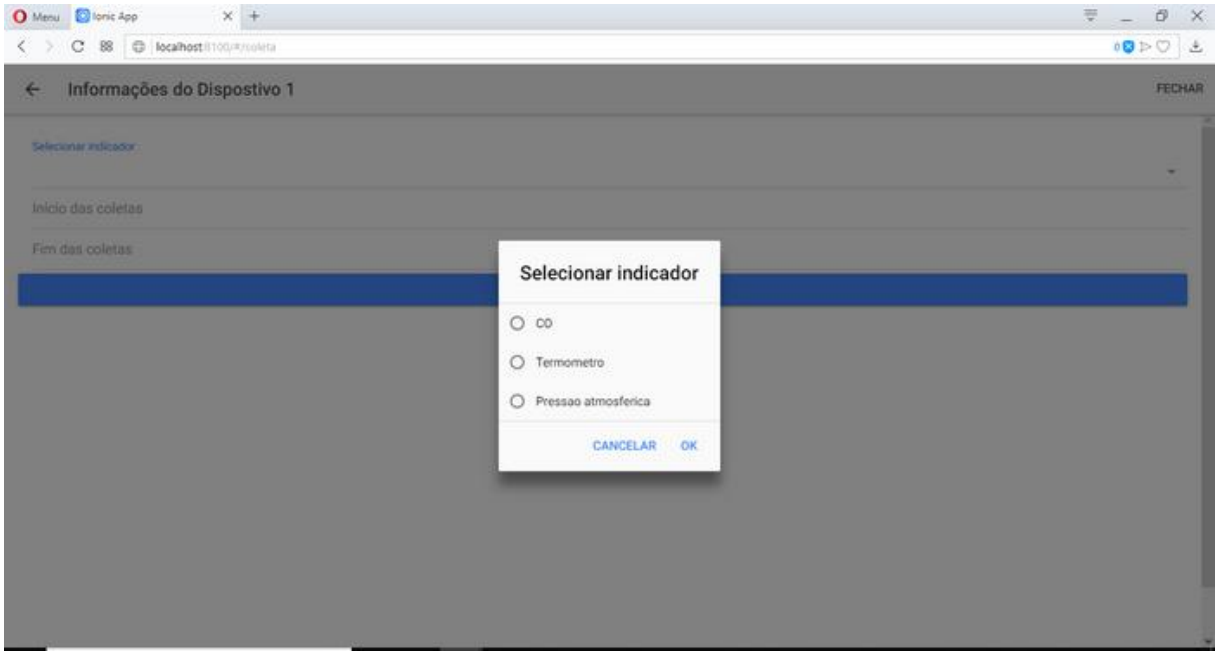
```
        }
    }
}

//Arquivo dispositivo.service.ts

import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { API_CONFIG } from "../../config/api.config";
import { DispositivoDTO } from "../../models/dispositivo.dto";
import { Observable } from "rxjs/Rx";

@Injectable()
export class DispositivoService {

    constructor(public http: HttpClient) {
    }

    findAll() : Observable<DispositivoDTO[]>{
        return this.http.get<DispositivoDTO[]>(`${API_CONFIG.baseUrl}/dispositivo/todos`);
    }
}

//Arquivo indicador.service.ts

import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { API_CONFIG } from "../../config/api.config";
import { Observable } from "rxjs/Rx";
import { IndicadorDTO } from "../../models/indicador.dto";

@Injectable()
export class IndicadorService {

    constructor(public http: HttpClient) {
    }

    findById(idIndicador : number) : Observable<IndicadorDTO>{
        return
this.http.get<IndicadorDTO>(`${API_CONFIG.baseUrl}/indicadores/${idIndicador}`);
    }
}

//Arquivo index.html

<!DOCTYPE html>
<html lang="en" dir="ltr">
<head>
  <meta charset="UTF-8">
  <title>Ionic App</title>
  <meta name="viewport" content="viewport-fit=cover, width=device-width, initial-scale=1.0,
minimum-scale=1.0, maximum-scale=1.0, user-scalable=no">
  <meta name="format-detection" content="telephone=no">
  <meta name="msapplication-tap-highlight" content="no">
```

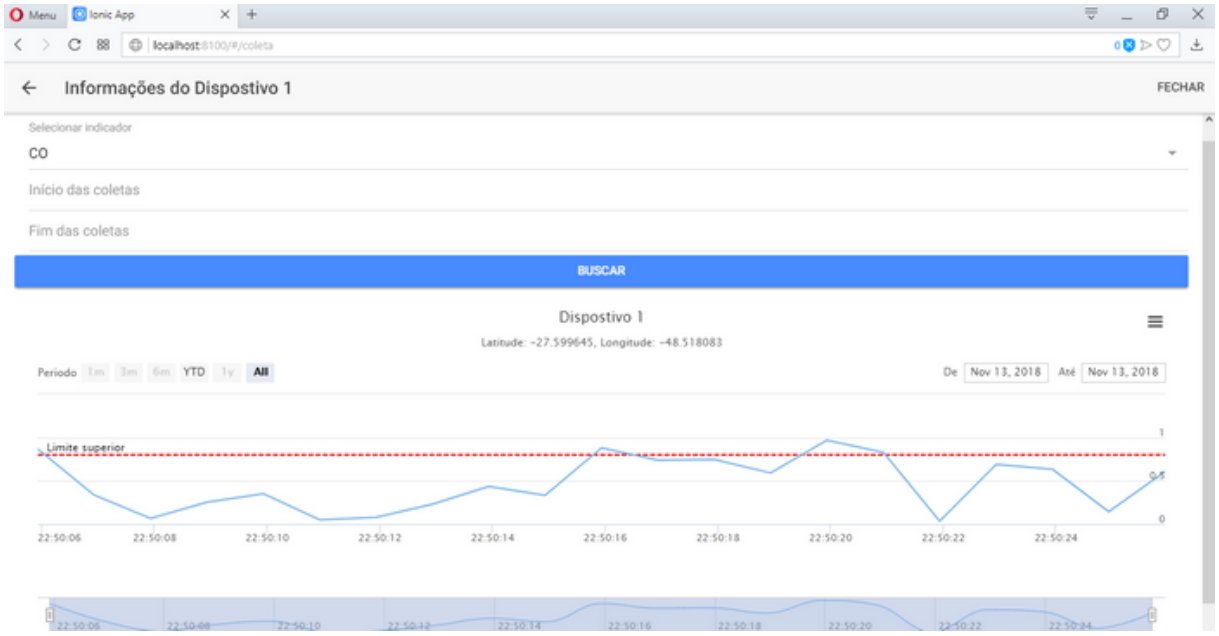
```
  <link rel="icon" type="image/x-icon" href="assets/icon/favicon.ico">
  <link rel="manifest" href="manifest.json">
  <meta name="theme-color" content="#4e8ef7">

  <!-- add to homescreen for ios -->
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black">

  <!-- cordova.js required for cordova apps (remove if not needed) -->
  <script src="cordova.js"></script>
  <script src="https://maps.googleapis.com/maps/api/js?key=<CHAVE>"></script>

  <!-- un-comment this code to enable service worker
  <script>
    if ('serviceWorker' in navigator) {
      navigator.serviceWorker.register('service-worker.js')
        .then(() => console.log('service worker installed'))
        .catch(err => console.error('Error', err));
    }
  </script>-->

  <link href="build/main.css" rel="stylesheet">

</head>
<body>

  <!-- Ionic's root component and where the app will load -->
  <ion-app></ion-app>

  <!-- The polyfills js is generated during the build process -->
  <script src="build/polyfills.js"></script>

  <!-- The vendor js is generated during the build process
       It contains all of the dependencies in node_modules -->
  <script src="build/vendor.js"></script>

  <!-- The main bundle js is generated during the build process -->
  <script src="build/main.js"></script>

</body>
</html>
```

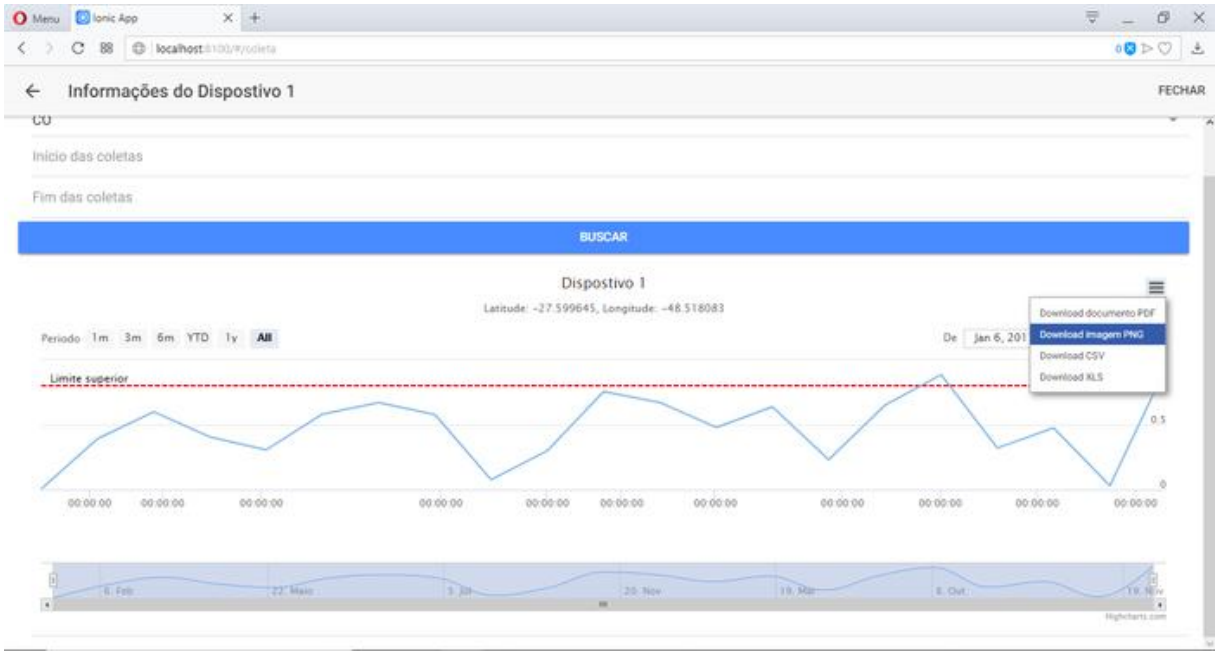
APÊNDICE D — Imagens da interface da Aplicação front-end






Menu
Ionic App
localhost:8100/#/coleta
Informações do Dispostivo 1
FECHAR
Selecionar indicador
Início das coletas
Fim das coletas
Selecionar indicador
CO
Termometro
Pressao atmosferica
CANCELAR
OK


Menu
Ionic App
localhost:8100/#/coleta
Informações do Dispostivo 1
FECHAR
Selecionar indicador
CO
Início das coletas
Fim das coletas
BUSCAR
Dispostivo 1
Latitude: -27.599645, Longitude: -48.518083
Periodo
YTD
All
De
Nov 13, 2018
Até
Nov 13, 2018
Limite superior
22:50:06
22:50:08
22:50:10
22:50:12
22:50:14
22:50:16
22:50:18
22:50:20
22:50:22
22:50:24


Menu
Ionic App
localhost
Informações do Dispostivo 1
FECHAR
CO
Início das coletas
Fim das coletas
BUSCAR
Dispostivo 1
Latitude: -27.599645, Longitude: -48.518083
Download documento PDF
Download imagem PNG
Download CSV
Download XLS
Período 1m 3m 6m YTD 1y All
De Jan 6, 201
Limite superior
0.5
0
00:00:00
Highcharts.com

APÊNDICE E — Artigo

Renovar: um MVP para monitorar a qualidade do ar

Francisco Sacco Flores Almeida Teixeira

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brazil

franciscosft@gmail.com

Abstract. *One of the segments of IoT is smartc ities, an environment with high innovative capacity, which uses data about the city to monitor it. Among the many areas used to consider an intelligent city, the environment is one of the most important, as it directly impacts the quality of life of the population. One of the factors that most impact on quality of life is air quality, which according to the WHO is considered the greatest environmental risk and causes more than 7 million deaths in 2016. In this way the work of completion of the course contemplates the development of an MVP, for the collection and monitoring of environmental data at IoT. This prototype is low-cost, collaborative and in the long run helps the development of an intelligent city.*

Resumo. *Um dos segmentos da IoT é cidades inteligentes, ambiente com alta capacidade inovativa, que usa dados sobre a cidade para monitorá-la. Dentre as muitas áreas utilizadas para considerar uma cidade inteligente, meio ambiente é uma das mais importantes, pois impacta diretamente na qualidade de vida da população. Um dos fatores que mais impactam na qualidade de vida é a qualidade do ar, que segundo a OMS é considerado o maior risco ambiental e causador de mais de 7 milhões de mortes somente em 2016. Dessa forma o trabalho de conclusão de curso contempla o desenvolvimento de um MVP, para a coleta e monitoramento de dados ambientais na IoT. Este protótipo é de baixo custo, colaborativo e que a longo prazo auxilie o desenvolvimento de uma cidade inteligente.*

1. Introdução

De acordo com Silva, Ferreira e Santos (2015) a problemática ambiental é uma preocupação para a comunidade científica. O crescimento da urbanização está atrelado a intensificação das atividades socioeconômicas, que conforme estudo da ONU metade da população mundial, em torno de 3,6 bilhões de pessoas, vivem em cidades. Silva, Ferreira e Santos (2015) também apontam que as áreas urbanas não planejadas sofrem demasiadamente pelo processo rápido de industrialização, assim como o grande fluxo de transporte urbano, que impacta diretamente no meio ambiente de uma cidade .

Florianópolis mostra bons indicadores socioeconômicos ao possuir, segundo IBGE, a terceira capital com a maior frota de carros per capita do país, porém essa mesma informação reflete um vetor de deterioração da qualidade do ar da cidade, visto que a maioria da frota é movida a queima de combustíveis fósseis. Conforme World Health Organization (2015), outro aspecto negativo, é a incidência de doenças causadas por poluição do ar, corroborando

para com isso mais uma pesquisa da World Health Organization (2016) informa que a poluição atmosférica já causou mais de 7 milhões de mortes somente no ano de 2016.

Algumas medidas já estão sendo tomadas para controlar os índices de poluição no Brasil e no mundo, tais como: promover políticas de desenvolvimento sustentável que apoiem a qualidade do ar saudável, divulgar causas de poluição atmosférica interna e externa, relacionar estas com doenças, apresentar os elevados custos das mortes provenientes da poluição atmosférica (WORLD HEALTH ORGANIZATION, 2015). Também para mitigar o problema de poluição atmosférica, o Brasil conta com a resolução da CONAMA nº03/1990. Ela, basicamente, apresenta quais são os níveis adequados dos poluentes e os limites de concentração que estes podem estar expostos no ambiente, para que não danem a saúde humana. Apesar da resolução estar em vigor há mais de 25 anos, Florianópolis sequer possui monitoramento da qualidade do ar.

Dessa forma o objetivo geral do trabalho é o desenvolvimento de um protótipo de coleta e monitoramento de dados referentes a qualidade do ar, disponibilizando estes em domínio público, por meio de uma solução IoT, por intermédio da pesquisa e implementação de uma solução de hardware, software livre, de baixo custo e colaborativo. Para contemplar a construção desse MVP, é necessário especificar o objetivo geral em três menores:

- Desenvolver um módulo para coletar e armazenar dados de sensores que medem a qualidade do ar;
- Desenvolver um módulo web para visualizar os relatórios a partir dos dados coletados.
- Propor uma documentação de um modelo escalável e colaborativo a partir dos módulos previamente desenvolvidos.

Este projeto foi, inicialmente, desenvolvido e testado como um MVP, aplicado em pequena escala dentro da UFSC, para posterior escala.

2. Desenvolvimento

Para atingir os objetivos elucidados anteriormente foi construído uma arquitetura de comunicação para compreender inteiramente o escopo do MVP do projeto Renovar, que vai englobar um sistema distribuído de sensores que se comunicam com um servidor da Internet, por meio do protocolo HTTP, enviando dados de monitoramento ambiental. Estes dados ficarão armazenados no servidor e também disponibilizados visualmente, de forma geo localizada, através de uma interface web, portanto nesta proposta, serão utilizadas várias tecnologias, distribuídas em diversos escopos:

- Protocolos de comunicação, sensores, projeto de hardware;
- Persistência e processamento de dados;
- Visualização de dados geo localizados.

A figura 1 representa o modelo da arquitetura de comunicação entre todos os componentes envolvidos no projeto: dispositivo de coleta de dados (coletores), servidor, *tablets*, *smartphones*, computadores tipo *desktop* e *notebooks*. Dentre os componentes apresentados é importante dividi-los em três grupos para compreender inteiramente a comunicação que será realizada: os coletores, o servidor e os clientes.

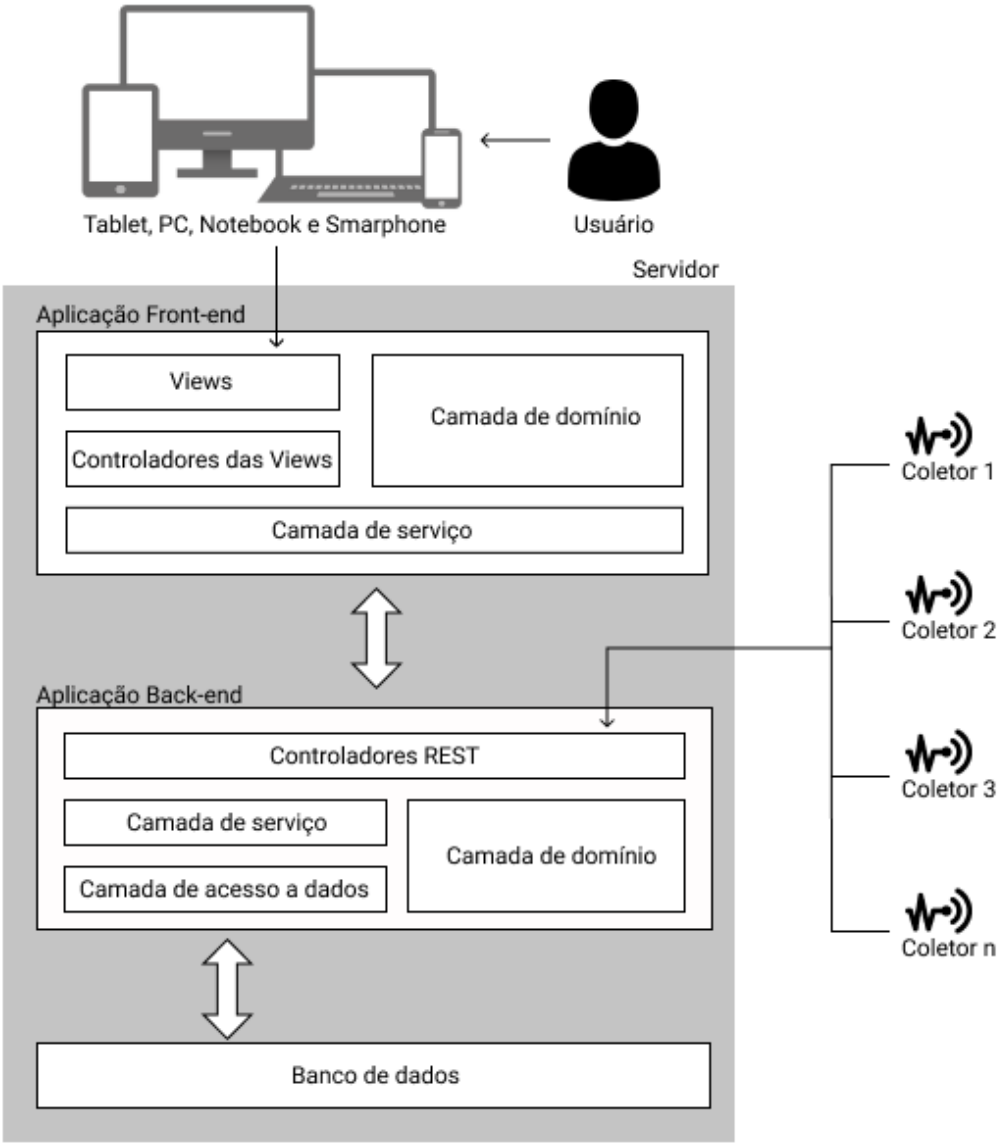


Figura 1. Arquitetura de comunicação

O primeiro grupo é composto pelos coletores, enumerados de 1 a “n”, que são os dispositivos de coleta que representam os nós de medição de qualidade do ar. Cada coletor será desenvolvido em parceria com o LCQAr, laboratório parceiro que auxiliou significamente na construção dos dispositivos. Desta forma os coletores advindos dessa parceria aproveitaram o arcabouço de dispositivos antigos aliados ao módulo de comunicação que desenvolvido para enviar os dados ao servidor, assim observa-se que um coletor é composto pelo seguintes módulos:

- O módulo de coleta, desenvolvido a partir da experiência e expertise do LCQAr, que possui os sensores para realizar as coletas de indicadores ambientais;
- O módulo de comunicação, que é composto por um sensor responsável por fazer conexão sem fio com a internet, realizando a comunicação com o servidor. Este módulo foi construído pelo o autor do presente trabalho, agregando valor aos antigos dispositivos de coleta do LCQAr.

O funcionamento geral de um coletor inicia com a captura dos dados ambientais por meio do módulo de coleta, estes dados são transportados internamente para o módulo de comunicação. No módulo de comunicação os dados são estruturados no formato JSON e após enviados para o servidor utilizando um recurso da API REST disponibilizado pela aplicação back-end.

O segundo grupo possui fisicamente apenas um componente: o servidor, contudo, logicamente ele é formado por três entidades: aplicação *back-end*, banco de dados e a aplicação *front-end*.

- A aplicação *back-end* é o elo de comunicação entre os coletores e o banco de dados, também entre a aplicação *front-end* e o banco de dados. O maior processamento da aplicação está nesta entidade do sistema, pois é nela que são feitos os processos de seleção, tratamento, filtros e inserção dos dados. A aplicação *back-end* é responsável por disponibilizar uma API REST, que permite que os coletores enviem as informações de dados ambientais para o banco de dados, assim como a aplicação *front-end* consulte os dados pertinentes a exibição na interface web. Também é importa dizer que a aplicação back-end foi modelada para medir qualidade do ar, contudo ela é flexível para abraçar outras grandezas e poluentes. A partir disso é possível adicionar outros tipos de sensores e desenvolver outras aplicações que assumam o papel da aplicação *front-end*.
- O banco de dados é responsável por armazenar todas as informações referentes aos coletores e os seus respectivos sistemas de medição, as coletas realizadas por eles, assim como quais os indicadores que são monitorados (concentração de poluentes, temperatura, pressão e outros parâmetros). Somente a aplicação *back-end* tem acesso ao banco de dados, desta forma só é possível inserir, modificar, deletar e ler dados por meio da API REST.
- A aplicação *front-end* é responsável pela comunicação dos usuários finais (clientes) com o servidor através de uma interface web exibindo dados geograficamente organizados. Para efetivar a comunicação com a aplicação back-end foi utilizado a API REST, a mesma consumida pelos coletores. A interface web exibe os dados de monitoramento coletados, a série histórica, assim como permite o download de relatórios.

O terceiro grupo é composto por “n” componentes, os clientes, sendo eles qualquer um dos enumerados a seguir: computadores tipo *desktop*, *notebook*, *smartphone* ou *tablet*.

Estes componentes estão nas mãos dos cidadãos, os usuários da plataforma Renovar, que tem acesso aos dados da concentração dos poluentes atmosféricos medidos por cada coletor. Os dados são exibidos na interface web da aplicação *front-end*, em forma de gráficos acessados a partir de uma coordenada disponibilizada sob mapa, como exemplificado na figura a seguir.

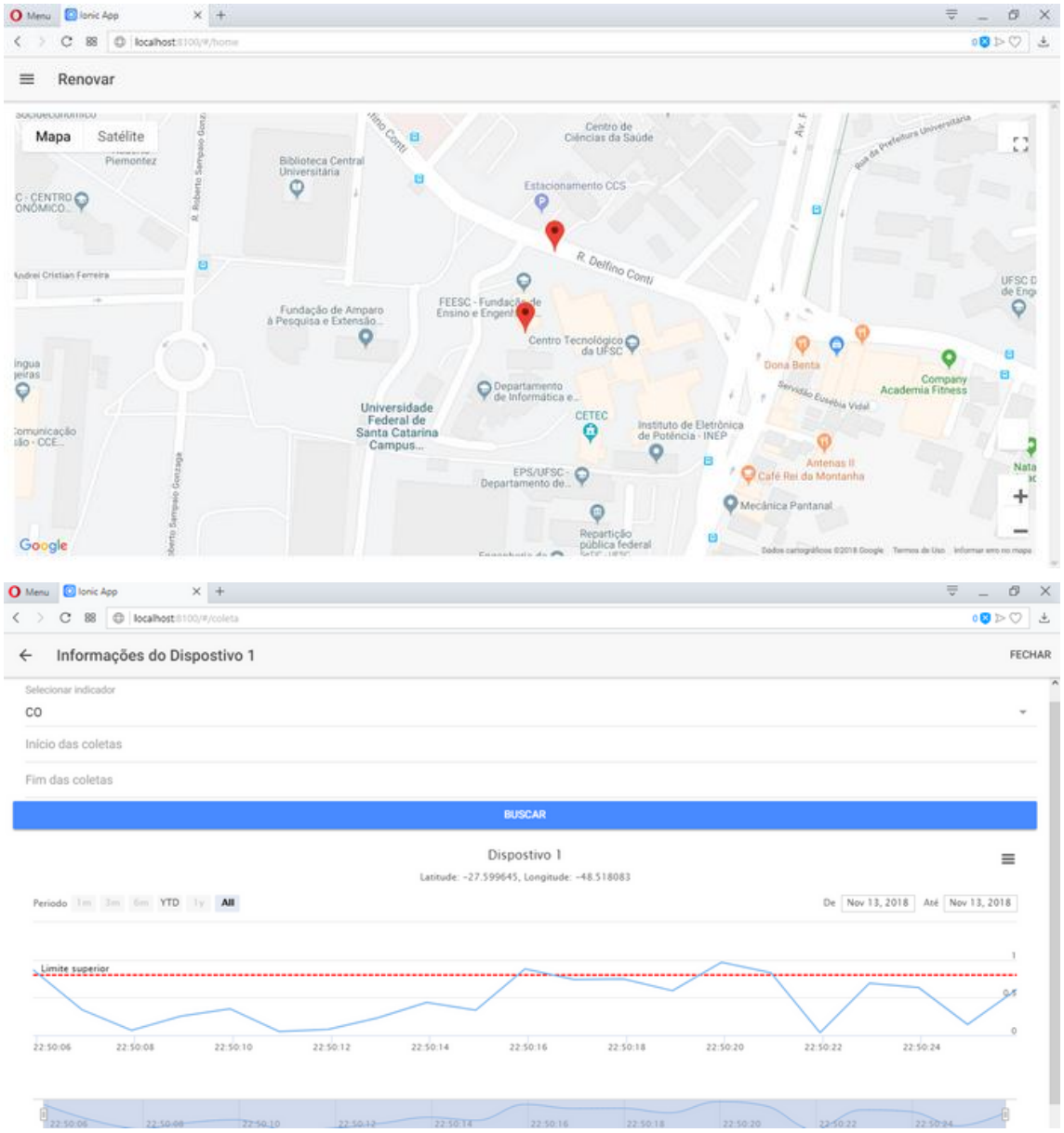


Figura 2. Interfaces da aplicação *front-end*

A partir da definição da arquitetura de comunicação foram utilizadas as seguintes tecnologias e ferramentas para implementar cada etapa:

- Módulo de comunicação do dispositivo de coleta: NodeMCU, Arduino IDE, linguagem de programação C++;
- Aplicação back-end: linguagem de programação Java, *framework Spring Boot,* brModelo, MySQL;
- Aplicacação front-end: framework Ionic, framework Angular, API Highcharts e Maps API JavaScript.

3. Conclusões

O primeiro objetivo atingido: o desenvolvimento do módulo para coletar e armazenar dados de sensores que medem a qualidade do ar. Este objetivo é considerado cumprido em função da:

- Implementação do banco de dados;

- Implementação da aplicação back-end que oferece a API REST Renovar;
- Implementação do módulo de comunicação do dispositivo de coleta;

Ainda é importante destacar que a aplicação back-end é generalista, ou seja, ela permite que outras grandezas ou poluentes sejam monitorados. Dessa forma o MVP da plataforma Renovar não fica restrito apenas a dados de qualidade do ar. Isto significa que a aplicação back-end é passível de escala e percebe-se que ela pode ser utilizada como agrupadora de diversos dados referentes ao meio ambiente.

O segundo objetivo que foi contemplado integralmente: o desenvolvimento de um módulo web para visualizar os relatórios a partir dos dados coletados. Esse objetivo tornou-se possível com a implementação da aplicação *front-end*, que funciona somente se a aplicação *back-end* estiver sendo executada. Percebe-se então que a aplicação *front-end* é dependente da *back-end*, contudo o contrário não se aplica.

As duas aplicações oferecem um alto valor agregado ao LCQAr, pois elas possibilitam uma economia de tempo e uma otimização no processo de monitoramento de dados ambientais que o laboratório faz. Uma vez que não há mais a necessidade de utilizar cartão SD nos dispositivos de monitoramento, nem o transporte do cartão para salvar as informações em um computador. A aplicação *back-end* oferece a API REST Renovar para o dispositivo de coleta enviar os dados, posteriormente serem armazenados no banco de dados. A aplicação *front-end* oferece a interface web para monitorar os dados e, também realizar o download deles quando necessário.

O terceiro e último objetivo que foi concluído: propor uma documentação de um modelo escalável e colaborativo a partir dos módulos previamente desenvolvidos. Desta forma o Trabalho de Conclusão de Curso assume a responsabilidade de ser o registro formal dessa documentação. O TCC registra todas as etapas de implementação explicando de forma geral cada uma delas, além disso nos apêndices do trabalho é disponibilizado o código fonte, que pode ser continuado, aperfeiçoado a partir de trabalhos futuros.

Referências

BRASIL. Ministério do Meio Ambiente. Resolução CONAMA n. 3 28 de junho de 1990. Diário Oficial da União, 22 de agosto de 1990. Disponível em:<http://www2.mma.gov.br/port/conama/legiabre.cfm?codlegi=100>. Acesso em:27 nov. 2018.

SILVA, Jadson Freire da; FERREIRA, Henrique dos Santos ; SANTOS, Marcelo Olímpio dos . Considerações sobre os estudos em clima urbano. GEAMA , Setembro 2015. Disponível em:<http://www.journals.ufrpe.br/index.php/geama/article/view/536>. Acesso em: 13 out. 2018.

WORLD HEALTH ORGANIZATION. Air pollution. World Health Organization. 2016. Disponível em: <http://apps.who.int/gho/data/node.sdg.3-9-viz-1?lang=en>. Acesso em: 26 nov. 2018.

WORLD HEALTH ORGANIZATION. Health and the Environment: Addressing the health impact of air pollution. 2015. Disponível

em:<http://apps.who.int/iris/bitstream/handle/10665/253206/A68_ACONF2Rev1-en.pdf?sequence=1&isAllowed=y>. Acesso em: 26 nov. 2018.

ANEXO A — Especificações técnicas do Coletor

No desenvolvimento do coletor foi utilizada a placa NodeMCU, que contém o Sistema-em-Chip (*SoC: System-on-Chip*) ESP8266 (Figura 1). O ESP8266 possui antena WiFi e stack TCP/IP embarcados, que potencializam sua utilização em aplicações relacionadas à Internet das Coisas. Além disso, o sistema consta de entradas digitais e interfaces serial UART, SPI e I2C que possibilitam seu uso também em aplicações de sensoriamento.

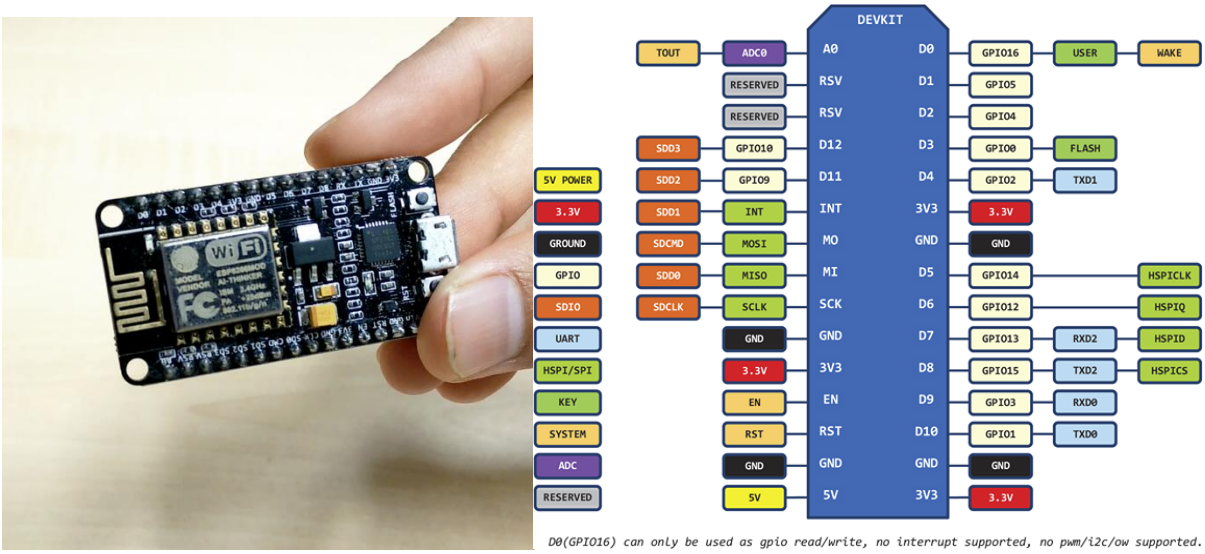


Figura 1. Placa NodeMCU (esquerda) e sua configuração de pinos (direita).

No caso específico da aplicação sendo desenvolvida, conforme se mostra na Figura 2, o NodeMCU se comunica via UART com um sensor de gás do fabricante *Spec Sensors*. O sistema coletor também inclui medição da pressão atmosférica via interface de comunicação I2C com o sensor BME280 da *Bosch Sensortech*. No mesmo barramento I2C, um Relógio de Tempo Real (*RTC: Real Time Clock*) mantém atualizadas a data e hora. Os dados coletados, além de serem transmitidos via WiFi até um servidor remoto, são armazenados em um cartão micro SD conectado à interface SPI do NodeMCU. O sistema todo é alimentado com uma tensão de 3.3 VDC.

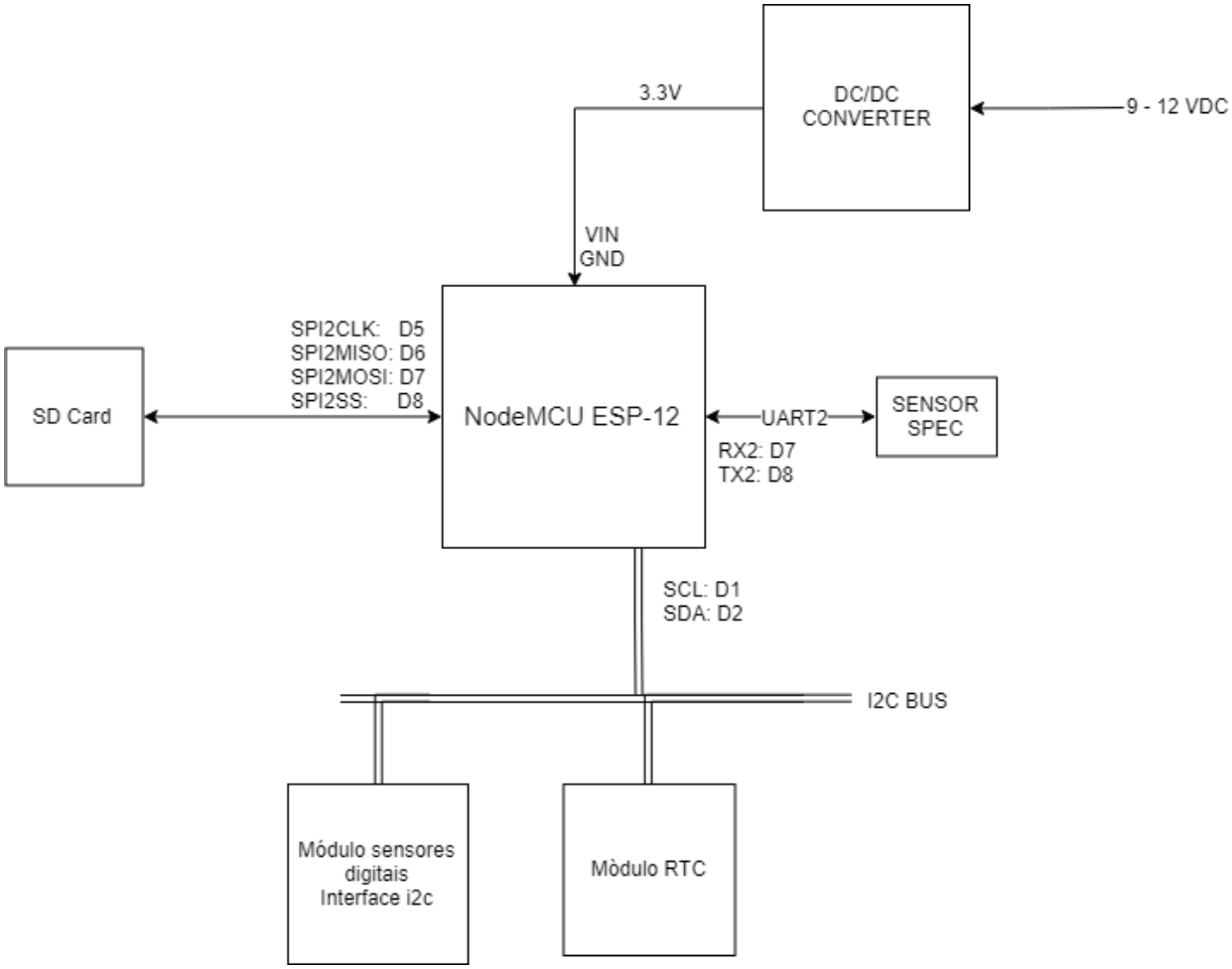


Figura 2. Diagrama em blocos de HW um coletor.

Os sensores de gases da Spec Sensors são transdutores eletroquímicos que geram uma corrente de saída, proporcional à concentração do gás alvo da medição. Estes sensores destacam-se por elevada precisão e baixa interferência cruzada. O processo de manufatura utilizado neste tipo de sensores, conhecido como Screen-Printed ElectroChemical, reduz os custos de fabricação e simplifica o processo de manufatura, e os sensores produzidos fornecem medições de qualidade com baixo consumo de energia (SPEC Sensors; https://www.spec-sensors.com/wp-content/uploads/2016/05/SPEC-Sensor-Operation-Overview.pdf). A Figura 3 mostra uma imagem do sensor de Dióxido de Nitrogênio desse fabricante modelo DGS-NO2 968-043.

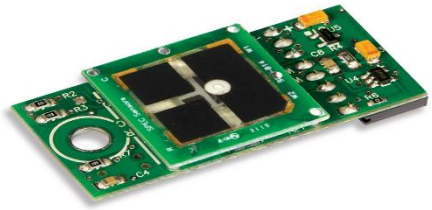


Figura 3. Sensor digital Spec DGS-NO2 968-043 para medição de Dióxido de Nitrogênio, disponível em https://www.spec-sensors.com/product/digital-gas-sensor-module-no2/

Junto com os sensores, o fabricante também providencia uma placa de condicionamento com microcontrolador, e sensor de temperatura e umidade relativa embarcados. Assim, a placa fornece as leituras de concentração de gás, temperatura e umidade relativa através de uma interface serial UART.