UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM ENGENHARIA DE SOFTWARE

BRUNO BARRETO FREITAS

UM FRAMEWORK PARA COLETA DE DADOS AMBIENTAIS EM CIDADES INTELIGENTES

QUIXADÁ

2017

BRUNO BARRETO FREITAS

UM FRAMEWORK PARA COLETA DE DADOS AMBIENTAIS EM CIDADES INTELIGENTES

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Orientador: Prof. Dr. Márcio Espíndola Freire Maia

QUIXADÁ

2017

BRUNO BARRETO FREITAS

UM FRAMEWORK PARA COLETA DE DADOS AMBIENTAIS EM CIDADES INTELIGENTES

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Márcio Espíndola Freire Maia (Orientador)
Universidade Federal do Ceará – UFC

Prof. Dr. Jefferson de Carvalho Silva
Universidade Federal do Ceará - UFC

Prof. Dr. Lincoln Souza Rocha
Universidade Federal do Ceará - UFC

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

F936f Freitas, Bruno Barreto.
Um framework para coleta de dados ambientais em cidades inteligentes / Bruno Barreto Freitas. – 2017.
49 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2017.
Orientação: Prof. Dr. Márcio Espíndola Freire Maia.

1. Framework. 2. Internet of things. 3. Computação móvel. I. Título.

CDD 005.1

A Deus.

Aos meus pais, Janilson Freitas e Rosa Maria, minha namorada Pamella Hayana e todos aqueles que sempre torceram por mim.

AGRADECIMENTOS

Primeiramente eu gostaria de agradecer a Deus, por ter me dado forças para a execução deste trabalho.

Aos meus pais Janilson Freitas e Rosa Maria, que sempre me deram todo o apoio necessário para que eu pudesse ter uma educação de qualidade e realizasse meu sonho de obter uma graduação nesta instituição de ensino.

A minha namorada Pamella Hayana, que sempre esteve ao meu lado, me apoiou e me incentivou durante toda a graduação, compartilhando comigo momentos de alegria e de tristeza que essa minha longa caminhada me proporcionou.

Ao meu orientador Prof. Dr. Márcio Espíndola Freire Maia, que sempre se mostrou disposto a me ajudar e teve toda a paciência necessária para que eu pudesse produzir este trabalho.

A todos os meus amigos que tive o prazer de conhecer durante esses anos na graduação, em especial Carlos Matheus, Jacques Nier, Amarildo Barros, Lucas Sales, Leonardo Brendo, Cayk Lima, Robson Cavalcante, Carlos Eduardo e Randerson Lessa por terem compartilhado comigo momentos únicos de alegria que ficarão marcados por toda minha vida.

“São nossas escolhas que mostram o que realmente somos, mais do que nossas habilidades”

(J.K Rowling)

RESUMO

A coleta de dados ambientais em cidades, como nível de CO_2, é muitas vezes feita de forma lenta e dispendiosa. Para solucionar este problema, surgiu o conceito de cidades inteligentes, que utiliza o paradigma de Internet das Coisas para solucionar este problema e outros que podem ser encontrados nas cidades sem a necessidade da intervenção humana. Com isso, vários trabalhos surgiram para solucionar este problema através da Internet das Coisas, compartilhando metodologias muito semelhantes através de etapas de coleta dos dados ambientais, armazenamento e disponibilização dos mesmos. A partir disso, este trabalho desenvolveu um *framework* para o desenvolvimento de aplicações de coleta de dados ambientais em cidades através do paradigma de IoT, englobando as etapas de coleta, armazenamento e disponibilização dos dados coletados. Neste trabalho também foi desenvolvida uma aplicação a partir do *framework* desenvolvido, no qual foi colocada em execução na cidade de Caucaia-CE com o intuito de validar o *framework*.

Palavras-chave: Framework. Internet das Coisas. Cidades Inteligentes

ABSTRACT

The collection of environmental data in cities, as CO_2 level, is done many times in a slowly and costly manner. To solve this problem, the concept of smart cities emerged, which uses the Internet of Things (IoT) paradigm to solve this problem and others that can be found in cities, without the need of human intervention. Thereby, several work emerged to solve this problem through the Internet of Things, sharing very similar methodologies through the steps of collecting, storage and providing environmental data. From this, this work developed a framework to the development of environmental data collection in cities applications through the IoT paradigm, encompassing the steps of collecting, storage and provision of the collected data. This work also developed an application from the developed framework, in which was put into execution at the city of Caucaia-CE, in order to validate the framework.

Keywords: Framework. Internet of Things. Smart Cities

LISTA DE FIGURAS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet of Things
MQTT	Message Queue Telemetry Transport
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
Pub/Sub	Publisher/Subscriber
GPRS	General Packet Radio Service

SUMÁRIO

1 INTRODUÇÃO

Estima-se que cerca de 3,2 bilhões de pessoas no mundo estejam conectadas à Internet (ONU, 2015). Muitos dados são trafegados pela rede, como conteúdos de mídia, e-mail, jogos online e notícias. Boa parte desses dados foram capturados e criados por seres humanos, ao tirar uma foto, capturar um áudio ou ao digitar um texto, por exemplo. O problema é que as pessoas possuem tempo e precisão limitada, o que significa que elas não garantem uma boa captura de dados sobre coisas do mundo real (ASHTON, 2009). Neste cenário, viu-se a oportunidade de se construir uma nova infraestrutura para a rede mundial de computadores, em que coisas ou objetos inteligentes do cotidiano interconectados possam interagir entre si a fim de coletar dados do mundo real sem a necessidade de uma intervenção humana e disponibilizá-las na Internet, paradigma conhecido por Internet das Coisas (IoT).

Segundo Atzori, Iera e Morabito (2010), a ideia básica de IoT é a presença de uma grande variedade de coisas ou objetos - como sensores, celulares, etc. - que através de esquemas de endereçamento único, são capazes de interagir e comunicar entre si, para alcançar algum objetivo específico. Estes objetos ou coisas são considerados inteligentes devido ao fato de possuir capacidades de computação e comunicação e integrar componentes eletrônicos. Com isso, novas oportunidades no setor de tecnologia da informação e comunicação (TIC) estão surgindo.

O uso da IoT é bastante amplo, podendo apresentar soluções em diversas áreas, desde a área da saúde até o monitoramento do meio ambiente. Com isso, pessoas comuns podem ter acesso às informações em tempo real através da Internet, como por exemplo, o estado de saúde de um determinado indivíduo, como pressão arterial e batimentos cardíacos, em que esses dados podem ser coletados através de uma simples pulseira com sensores, disponibilizando esses dados na Internet ou para um determinado usuário final, ou informações de como está a qualidade do ar de uma determinada área em que o indivíduo se encontra.

Uma das áreas de destaque da IoT são as chamadas cidades inteligentes. Uma cidade inteligente pode ser definida como a aplicação de IoT em um contexto urbano, que adota soluções de comunicação e informação para o gerenciamento de assuntos públicos (ZANELLA et al., 2014). Esse gerenciamento é feito através de um monitoramento da cidade avaliando dados relacionados a diversos fatores, que possam afetar a saúde e o bem estar da população, de forma eficiente (JIN et al., 2014).Com isso, um dos principais serviços que podem ser disponibilizados por uma cidade inteligente é a coleta de dados ambientais, como o nível de poluição de um

determinado espaço público, a temperatura e a umidade em tempo real da cidade, nível de barulho em locais público ou em ruas movimentadas entre outros fatores.

A motivação para a utilização de IoT em monitoramento de cidades em relação a coleta de dados ambientais se dá pelo fato de que há poucas cidades com plataformas ou sistemas para monitoramento das mesmas em tempo real. A maioria das cidades utilizam a estratégia de coleta de dados, análise offline e ação, seguido por alguns ajustes necessários no sistema e a repetição do processo inteiro novamente. O problema desta estratégia é que a coleta de dados realizada desta forma normalmente é muito cara e difícil de se replicar. Com isso, existe a demanda por tecnologias inteligentes que realizem este processo em tempo real, aplicando-se então o conceito de IoT nas cidades (JIN et al., 2014).

Atualmente, vários trabalhos já foram realizados apresentando uma solução para coleta de dados ambientais em cidades inteligentes, porém não através de uma proposta de um *framework*. Portanto, este trabalho identificou uma forma de disponibilizar um *framework* para que o desenvolvimento de aplicações deste gênero pudessem ser desenvolvidas de forma mais genérica.

Este trabalho tem como objetivo a construção de um *framework*, que busca oferecer uma estrutura genérica para o desenvolvimento de aplicações de coleta de dados ambientais em cidades através do paradigma de IoT.

2 FUNDAMENTAÇÃO TEÓRICA

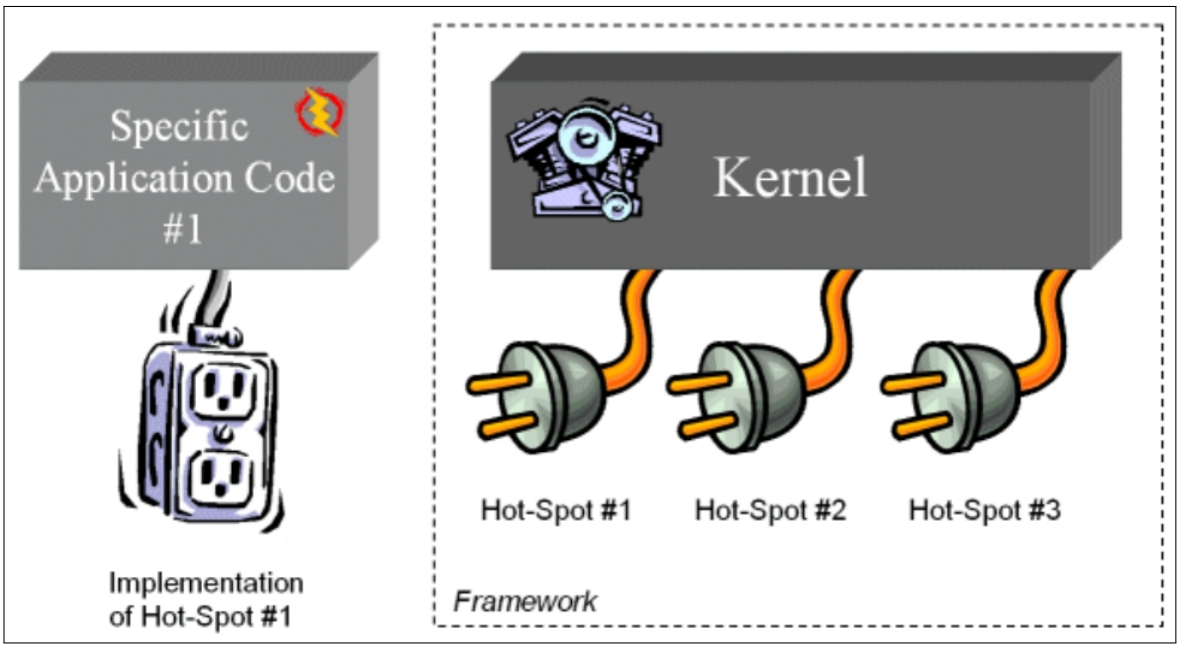
2.1 *Framework*

Segundo Johnson e Foote (1988), um *framework* é um conjunto de classes abstratas que colaboram entre si, que representam um design abstrato, que é utilizado para a construção de uma família de aplicações semelhantes, focando no reuso. Um *framework* é uma aplicação quase completa, necessitando de configurações e de implementações em certos pontos que serão específicos para uma aplicação específica do domínio do mesmo. *Frameworks* representam um alto nível de reusabilidade de software, fazendo com que padrões arquiteturais possam ser reusados em aplicações, podendo melhorar significativamente a qualidade das mesmas (WOLFGANG, 1994).

Um *framework* é muitas vezes confundido com outros conceitos, como padrão de projetos e aplicações orientadas a objetos. Diferencia-se de padrões de projetos pelo fato de que um *framework* é específico a um domínio de aplicação, enquanto padrões de projetos podem ser utilizados em qualquer tipo de aplicação. Em relação a uma aplicação orientada a objetos, diferencia-se pelo fato de uma aplicação orientada a objetos apresentar uma implementação completa e executável de uma determinada especificação, enquanto *frameworks* captura a funcionalidade da aplicação e não é executável por não cobrir inicialmente o comportamento específico da aplicação desejada (AREVALO, 2000).

Um *framework* apresenta pontos flexíveis que devem ser alterados para o funcionamento de uma determinada aplicação e pontos que são geralmente imutáveis e de difícil alteração. Os pontos flexíveis de um *framework* são conhecidos como *hot spots*, que são classes abstratas ou métodos que devem ser implementados para que uma aplicação executável possa ser gerada a partir do *framework*. Já os pontos imutáveis são conhecidos como *frozen spots*, que ao contrário dos *hot spots*, são classes ou pedaços de código já implementados, que representam geralmente o núcleo da aplicação que irá ser gerada e são responsáveis por utilizar os *hot spots* implementados (MARKIEWICZ; LUCENA, 2001).

Figura 1 – Representação de um *framework* em relação ao seus *hot spots* e *frozen spot*



Fonte – Markiewicz e Lucena (2001)

Este conceito foi utilizado neste trabalho para o desenvolvimento de um *framework* para o desenvolvimento de aplicações IoT para coleta de dados ambientais em cidades inteligentes.

2.2 Internet das Coisas

Embora existam várias definições de IoT, a ideia básica de uma rede de coisas ou objetos interconectados que coletam informações do meio ambiente e interagem com o mundo físico, utilizando padrões de Internet para promover serviços de informações para transferências, análises, aplicações e comunicações continua a mesma (GUBBI et al., 2013).

Este paradigma dará uma visão diferente a internet tradicional, em que objetos inteligentes interconectados formarão ambientes de computação difusivos (MIORANDI et al., 2012). Segundo Miorandi et al. (2012), a IoT é construída sobre três pilares, em relação a habilidade que objetos inteligentes têm de serem identificáveis, se comunicarem e interagirem. Tais objetos podem ser definidos como entidades que:

- Possuir um conjunto de características físicas, como tamanho, largura etc.
- Possuir um mínimo de funcionalidades de comunicação, como receber e enviar mensagens.
- Possuir um identificador único.

- Ser associado ao menos a um nome que seja legível para os humanos, que apresenta uma descrição do objeto e a razão do seu propósito.
- Possuir algumas capacidades de computação básica, como a habilidade de poder enviar e receber mensagens e algumas funcionalidades um pouco mais complexas, como gerenciamento de tarefas de rede etc.
- Possuir capacidade de sentir fenômenos físicos, como temperatura, umidade, qualidade do ar, nível do sonoro do ambiente etc.

Segundo Miorandi et al. (2012), as características principais que o paradigma de IoT precisa dar suporte são:

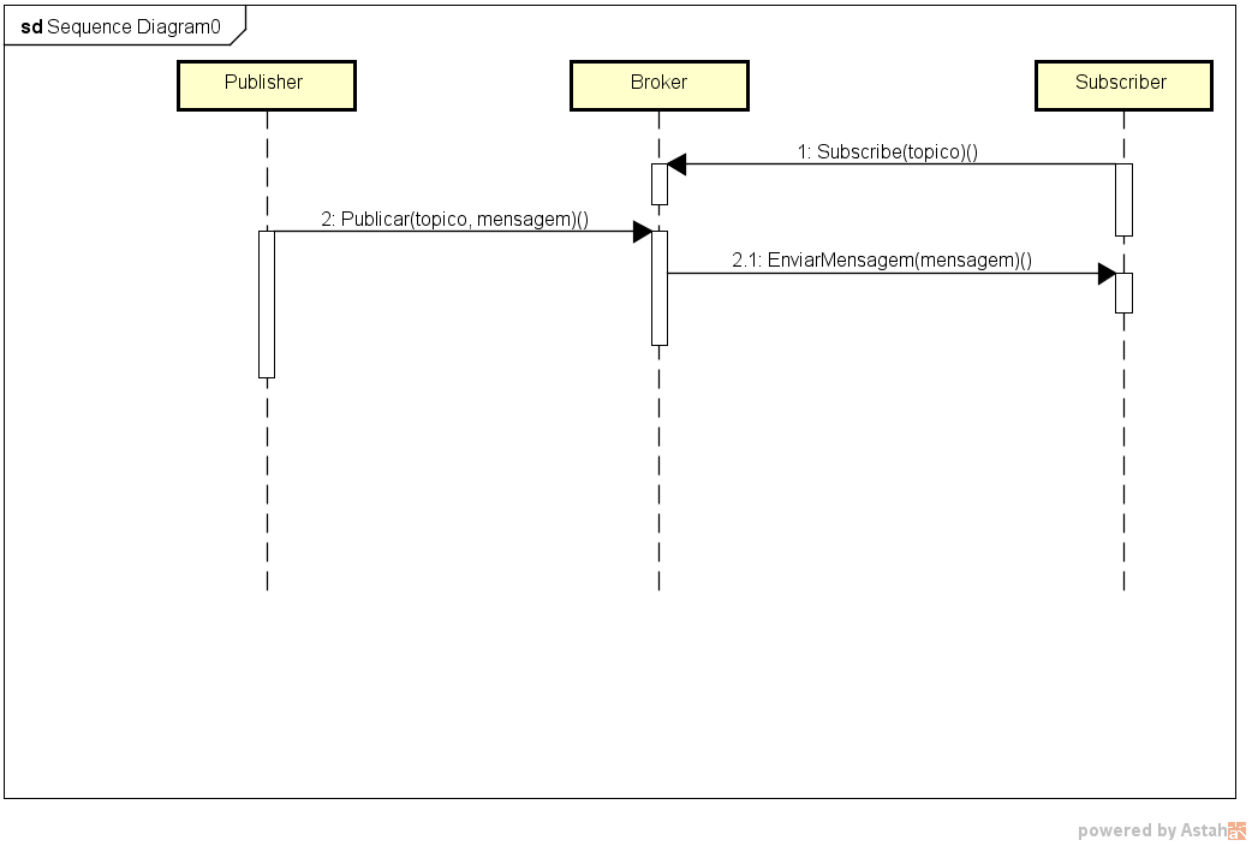
- Heterogeneidade de dispositivos, já que IoT apresenta uma grande variedade de dispositivos, que devem apresentar várias capacidades diferentes em termos de computação e comunicação.
- Escalabilidade, já que muitas questões surgem quando objetos se conectam a uma rede global de informações, como a questão da comunicação de dados, já que existe uma grande quantidade de interconexões entre os objetos inteligentes.
- Troca de dados ubíquos através de tecnologias sem fio, que fará com que objetos inteligentes se conectem em uma rede.
- Soluções de optimização de energia, já que a energia é um dos recursos que deve ser dado mais atenção em IoT, pois quanto mais tempo os objetos inteligentes estiverem ativos, mais informações poderão ser coletadas.
- Localização e capacidades de rastreamento, já que é de extrema importância rastrear a localização de objetos no domínio físico, para que se saiba o local exato em que os dados foram coletados pelo objeto inteligente.
- Capacidade de auto organização, é a habilidade que objetos inteligentes têm de reagirem de forma autônoma a várias situações, com o intuito de minimizar a intervenção humana.
- Interoperabilidade semântica e gerenciamento de dados, pois uma grande quantidade de dados será gerada, com isso, faz-se necessário tornar esses dados em informações úteis e legíveis para uso posterior.

A IoT pode utilizar vários protocolos de rede, como o protocolo *Message Queue Telemetry Transport (MQTT)*. O protocolo MQTT é projetado especialmente para operações de baixo custo e para dispositivos com largura de banda e energia limitada, no qual utiliza

o padrão arquitetural *publisher/subscriber*(pub/sub) (MQTT, 2017). O princípio do modelo de comunicação (pub/sub) é ter componentes interessados em consumir certas informações produzidas por outro componente. Os componentes que produzem as informações são chamados *publishers* e os que consomem as informações *subscribers*. Já a entidade que garante que os dados produzidos por um *publisher* chegue até um *subscriber* se chama *broker* (HUNKELER; TRUONG; STANFORD-CLARK, 2008).

Segundo Hunkeler, Truong e Stanford-Clark (2008), existem três tipos de sistemas pub/sub: baseados em tópicos, baseado em tipo e baseado em conteúdo. No baseado em tópicos, em que a comunicação entre os *subscribers* e os *publishers* é feita através de um conjunto de tópicos conhecidos, que funcionam como um canal de comunicação. No baseado em tipo, o *subscriber* define o tipo do dado em que está interessado em receber, como por exemplo temperatura. Já o baseado em conteúdo, o *subscriber* descreve o conteúdo de mensagens que quer receber, como por exemplo, apenas as mensagens contendo temperatura e luz, em que a temperatura esteja abaixo de um certo valor e que a luz esteja acesa.

Figura 2 – Pub/Sub baseado em tópico



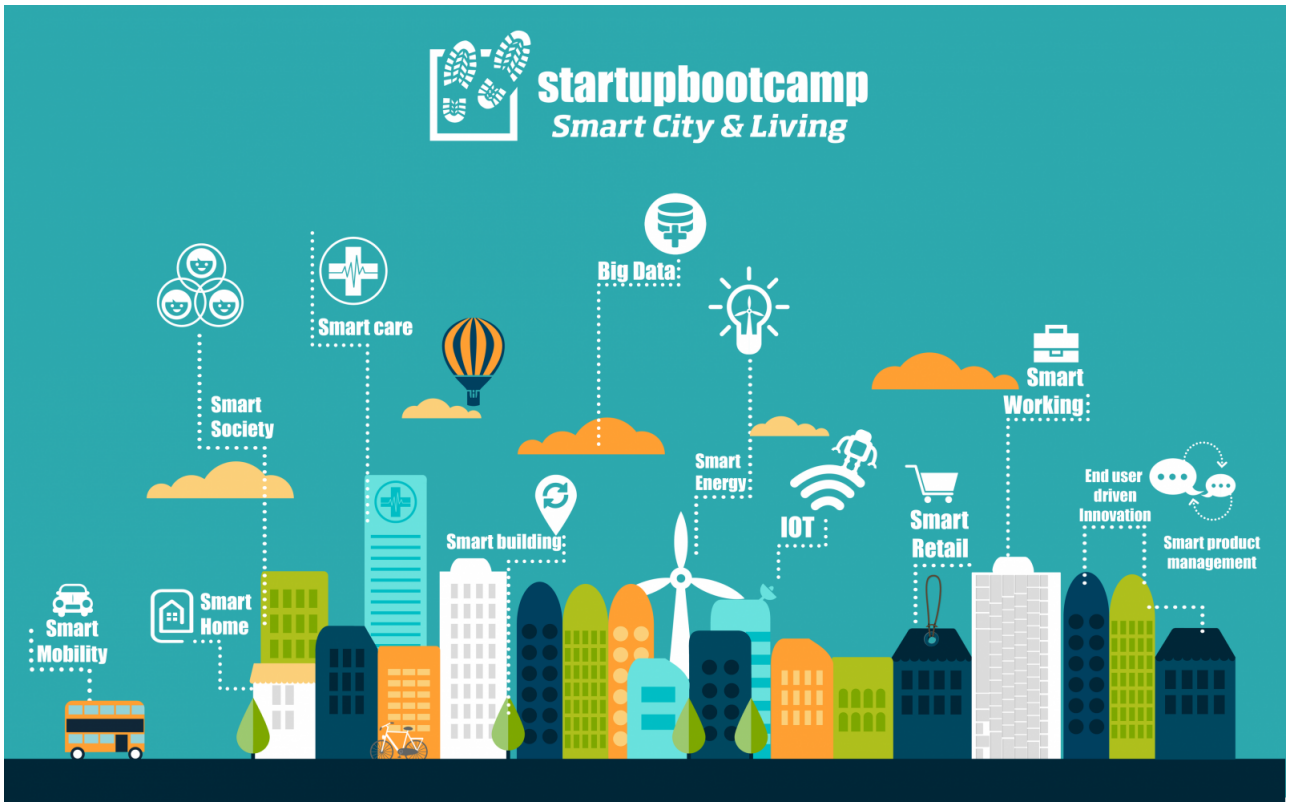
Fonte: Elaborada pelo autor

O conceito de Internet das Coisas foi utilizado como o paradigma do *framework* desenvolvido, em que um objeto inteligente coleta informações do meio físico e utiliza a Internet para o envio desses dados, utilizando o protocolo MQTT baseado em tópicos.

2.3 Cidades Inteligentes

Uma cidade inteligente tem como objetivo explorar as mais avançadas tecnologias de comunicação para dar suporte a serviços que agreguem valor para a administração da cidade e para os cidadãos. Uma cidade inteligente é realizada graças ao paradigma de IoT, que promove fácil acesso a diversos dispositivos como câmeras de segurança, sensores de monitoramento, veículos etc. Com isso, uma enorme quantidade de dados é gerada, promovendo vários serviços para os cidadãos, companhias e administrações públicas (ZANELLA et al., 2014).

Figura 3 – Visão de uma cidade inteligente



Fonte: startupbootcamp (2016)

Segundo Zanella et al. (2014), vários serviços de monitoramento podem ser promovidos através de uma cidade inteligente, como:

- Saúde estrutural de construções, em que dados como a integridade estrutural

de construções são coletados através de sensores e distribuídos através de uma base de dados distribuída. Com isso, deverá reduzir a necessidade de testes caros realizados periodicamente por operadores humanos e também poderá alertar a população sobre a necessidade de cuidar de patrimônios históricos da cidade.

- Gerenciamento de lixo, em que containers de lixo inteligentes detectam a sua quantidade de lixo, poderiam melhorar as rotas dos caminhões responsáveis por sua coleta ao indicar os locais em que apresentam containers com maior concentração de lixo.
- Qualidade do Ar, em que o paradigma de IoT poderia mensurar o nível de gases poluentes como CO_2 nas cidades, através de objetos inteligentes equipados com sensores apropriados.
- Monitoramento de Barulho, em que objetos inteligentes poderiam coletar o nível sonoro de locais da cidade periodicamente, auxiliando a administração pública e informando a população sobre os locais que apresentam barulho acima do permitido na cidade.
- Iluminação inteligente, em que a iluminação da cidade seria optimizada de acordo com período do dia e condições climáticas, gerando uma economia de energia.

Uma cidade inteligente é baseada em uma arquitetura centralizada, em que vários objetos ou coisas inteligentes espalhadas na cidade geram uma grande quantidade de dados diferentes que são enviadas através de tecnologias de comunicação para um servidor central, onde serão processados e armazenados (ZANELLA et al., 2014). Segundo Su, Li e Fu (2011), a arquitetura de uma cidade inteligente pode ser representada através de três camadas:

- Camada de percepção, em que os objetos inteligentes realizam a coleta de dados através de sensores, câmeras, GPS etc.
- Camada de rede, em que os dados coletados na camada de percepção são enviados para um determinado local, um servidor central por exemplo, através de tecnologias de comunicação, como Wireless e 3G.
- Camada de aplicação, em que os dados coletados são analisados e processados através de computação em nuvem ou outras tecnologias inteligentes.

Esse conceito foi utilizado como objetivo das aplicações desenvolvidas a partir do *framework* desenvolvido no trabalho, ao coletar dados ambientais em cidades inteligentes.

2.4 Arduino

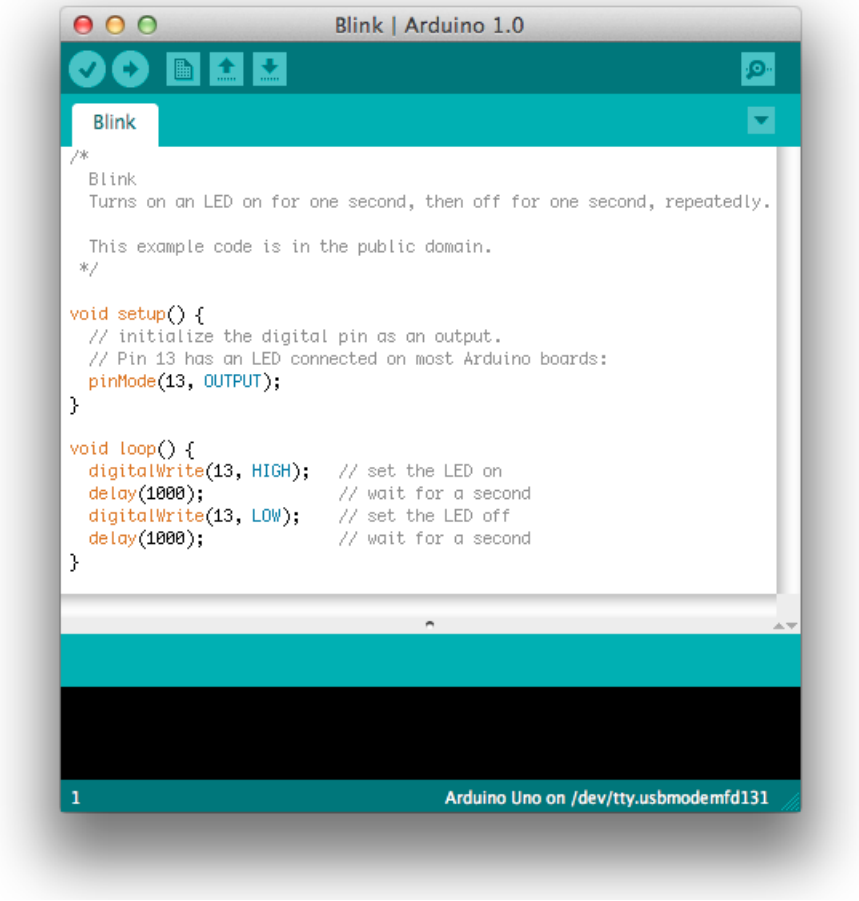
Arduino é uma plataforma de prototipagem de código aberto de fácil utilização, integrando hardware e software de maneira simples, que pode ler entradas de diversas fontes, como sensores de luz e de movimento, ou até de uma mensagem do Twitter e tornar isso em uma saída, podendo ativar um LED ou colocar dados online (ARDUINO, 2016). O Arduino pode ser utilizado em diversos cenários, como no desenvolvimento de um objeto iterativo e independente ou na conexão à Internet, enviando dados coletados por sensores para um site para que possam ser exibidos através de gráficos. O Arduino ainda pode utilizar shields, que são placas de circuito que contêm dispositivos como GPS, GPRS, display de LCD e etc, para obter mais funcionalidades (MCROBERTS, 2015).

O Arduino é bastante indicado para estudantes e pessoas interessadas em aprender sobre o assunto, por facilitar o processo de trabalhar com microcontroladores (ARDUINO, 2016). O Arduino apresenta algumas características que tornam o Arduino atrativo, são elas:

- O Arduino possui um baixo custo, o que incentiva o seu uso para os iniciantes na área.
- A IDE do Arduino é multi-plataforma, podendo rodar em ambientes Windows, Macintosh OSX e Linux.
- O Arduino apresenta um ambiente de programação simples, baseado no ambiente de programação procedural, que é bastante familiar para programadores iniciantes.
- O Arduino apresenta um software e hardware extensível, em que a linguagem pode ser expandida através de C++ e o hardware pode ser modificado por designers de circuitos experientes, possibilitando-os a fazer as suas próprias versões do módulo.

Para programar o Arduino, é necessário utilizar a sua IDE, um software multi-plataforma que roda em ambientes Windows, Macintosh OSX e Linux, que pode ser baixado de forma gratuita no site oficial da plataforma. A linguagem padrão utilizada pelo Arduino é baseada em C/C++ (MCROBERTS, 2015).

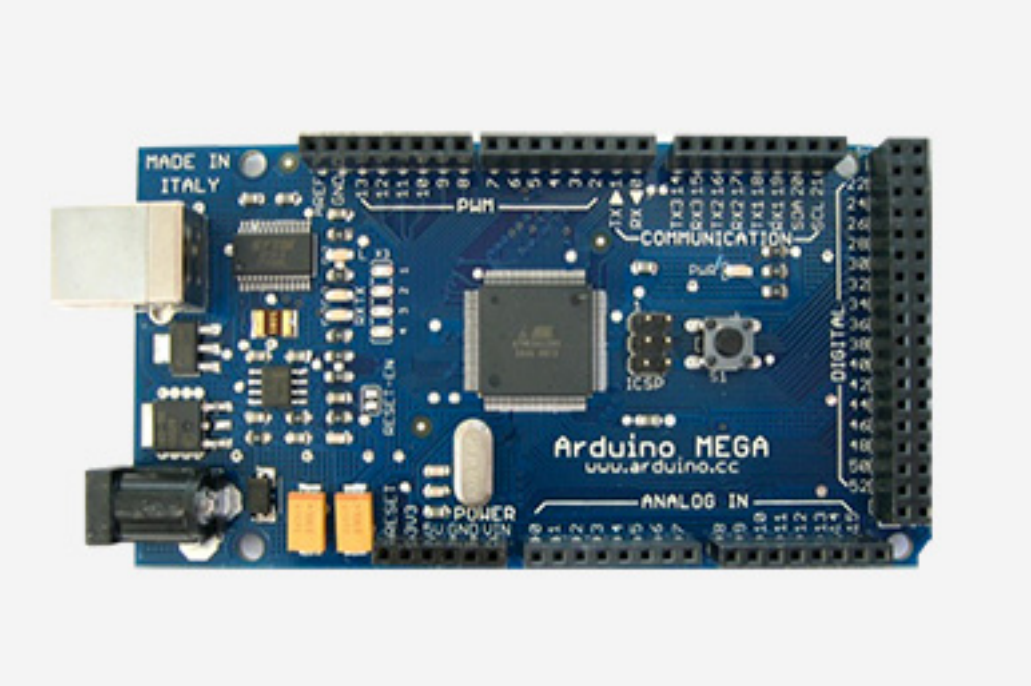
Figura 4 – IDE do Arduino



Fonte: Arduino (2016)

O Arduino é encontrado em algumas versões diferentes, em que como por exemplo a versão MEGA. Segundo MCROBERTS (2015), o Arduino MEGA é bastante indicado para projetos que necessitem de um processamento melhor, já que esta versão do Arduino conta com um processador mais avançado e com uma memória maior do que a versão UNO, a mais simples do Arduino.

Figura 5 – Arduino MEGA



Fonte: Arduino (2016)

O Arduino foi utilizado nesse trabalho como o dispositivo responsável por realizar a coleta de dados ambientais no *framework* proposto neste trabalho.

2.5 *REST*

O REST (Representational State Transfer) é um estilo arquitetural que tem como princípio representar e manipular informações sobre recursos. Esses recursos são identificados através do padrão universal URIs e são manipulados através de uma interface padronizada HTTP, que é feita através de um serviço web RESTful, em que o seu acesso é realizado da mesma maneira como a de um Website (MENG; MEI; YAN, 2009).

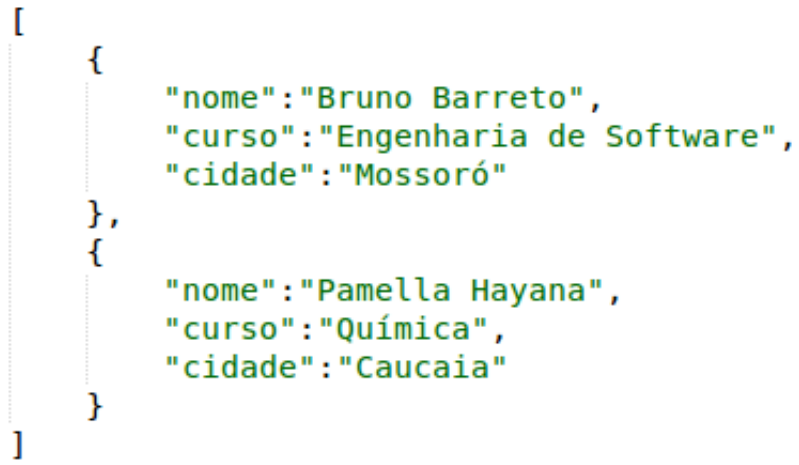
Segundo Meng, Mei e Yan (2009), o estilo arquitetônico do REST tem as seguintes características e vantagens:

- Endereçamento, em que os recursos são marcados e disponíveis na Web através de URIs, que será o seu identificador global.
- Links e conectividade, em que recursos são lincados um ao outro por hiperlinks.
- Sem estado, em que cada requisição inclui todas as informações necessárias para que os servidores entendam.

- Interface uniforme, em que os recursos são manipulas através de quatro operações HTTP (GET, PUT, DELETE e POST). A operação POST é responsável por criar um novo recurso e DELETE por excluí-lo. A operação GET é responsável por retornar um determinado recurso e PUT por atualizá-lo.

O REST utiliza alguns padrões de texto para a transferência de dados, como JSON. Segundo Crockford (2006), "JavaScript Object Notation (JSON) é um formato de texto para a serialização de dados estruturados". O JSON é projetado para ser uma linguagem de troca de dados que seja facilmente legível por seres humanos e de fácil manipulação por computadores.

Figura 6 – Exemplo de um array JSON



Fonte: Elaborada pelo autor

Este conceito foi utilizado nesse trabalho na implementação de um *Web Service* responsável pela disponibilização das coletas realizadas utilizando o *framework*.

2.6 Android

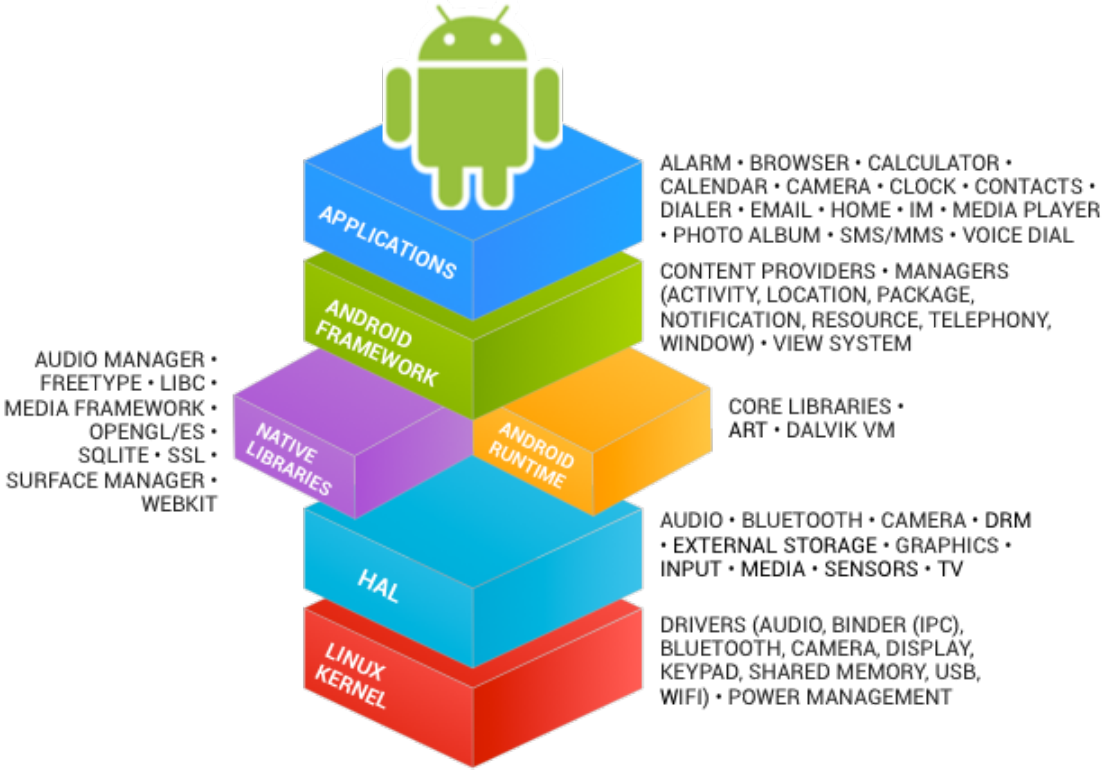
Android é um sistema operacional voltado para dispositivos móveis de código aberto, baseado no kernel 2.6 do Linux (LECHETA, 2013). É um dos sistemas operacionais mais utilizados no mundo, alcançando mais de um bilhão de dispositivos, como Smartphones, Tablets, TVs etc (ANDROID, 2016).

O Android possui um conjunto de aplicações padrões como um cliente e-mail, mapas, navegador web, contatos entre outros, que podem ser utilizadas no desenvolvimento de outras aplicações. Todas as aplicações são escritas utilizando a linguagem Java (DEVELOPERS,

2011).

O Android apresenta uma arquitetura muito bem definida (Figura 2)

Figura 7 – Arquitetura do Android



Fonte: Android Source (2016)

A arquitetura do Android é representada através de uma pilha, em que a medida que descemos na pilha, menos abstração teremos.

No topo da pilha temos os aplicativos, em que são ferramentas utilizadas pelo usuário para um determinado fim. O framework Android é responsável por dar acesso as bibliotecas nativas do Android. O Android Runtime compreende as bibliotecas nativas do Android, responsáveis por funções básicas como acesso a base de dados e renderização de imagens. A camada HAL da pilha apresenta os componentes físicos do dispositivo em que o Android está presente, como Bluetooth, Câmera, Sensores entre outros. Por último, o Kernel Linux é responsável pelo gerenciamento de memória, comunicação com os componentes físicos do dispositivo, acesso à rede entre outros.

O Android foi utilizado no desenvolvimento de uma aplicação móvel responsável por mostrar as coletas realizadas pela aplicação desenvolvida a partir do *framework* proposto.

3 TRABALHOS RELACIONADOS

Nesta seção, serão apresentados brevemente trabalhos que apresentam o contexto do *framework* a ser desenvolvido neste trabalho, apresentando suas semelhanças e diferenças no que pretende ser desenvolvido.

Mendez et al. (2011) apresenta uma solução de coleta de dados ambientais para monitorar a poluição do ar em cidades. Para isto, foi desenvolvido um dispositivo através de sensores integrados em uma placa. Este dispositivo utilizou sensores de monóxido e dióxido de carbono, gás combustível, temperatura e umidade relativa do ar. Ao coletar os dados provenientes dos sensores, os mesmos são enviados para um celular através de uma conexão *bluetooth* realizada por um módulo Arduino, onde são processados e enviados para um servidor Web, juntamente com a posição GPS, onde são armazenados em uma base de dados, em que posteriormente uma aplicação Web também desenvolvida realiza a amostragem dos dados em um mapa.

A proposta do *framework* a ser desenvolvido neste trabalho assemelha-se ao trabalho apresentado em Mendez et al. (2011), já que o seu contexto foi relacionado a coleta de dados ambientais. Além disso, o trabalho utiliza uma metodologia em que os dados são coletados, processados e enviados a um servidor para que possam ser armazenados e disponibilizados, a mesma metodologia na qual o *framework* irá utilizar.

Entretanto, este trabalho diferencia-se ao de Mendez et al. (2011) por apresentar um *framework* em que o processamento e o envio dos dados coletados para o servidor sejam realizados através de um dispositivo capaz de realizar esta funcionalidade e que seja utilizado através de um Arduino, ao contrário de Mendez et al. (2011) que utiliza um celular para que esta funcionalidade seja realizada.

Vagnoli et al. (2014) apresenta uma solução para monitoramento do meio ambiente urbano, ao coletar dados ambientais relacionados a poluição do ar, temperatura e umidade. Para isto, foi desenvolvido uma placa de circuito, equipado com sensores de CO, CO_2, O_3, NO_2, CH_4, Temperatura, Umidade e Barulho e um microcontrolador para que esses dados possam ser lidos e processados. Além disso, este dispositivo é equipado com um dispositivo baseado na tecnologia GPRS, que será responsável pelo envio dos dados coletados para um servidor Web, onde serão armazenados em uma base de dados. Posteriormente, uma aplicação Web foi desenvolvida para a amostragem dos dados em um mapa.

A proposta do *framework* a ser desenvolvido neste trabalho assemelha-se ao trabalho de Vagnoli et al. (2014) por utilizar a mesma metodologia, em que os dados são coletados,

processados e enviados através de dispositivos de conexão com a Internet.

Entretanto, este trabalho diferencia-se ao de Vagnoli et al. (2014) pelo fato do *framework* a ser desenvolvido utilizar dispositivos Arduino, enquanto Vagnoli et al. (2014) utiliza um placa de circuito própria para a realização da coleta dos dados.

Em Devarakonda et al. (2013), foi desenvolvida uma solução para coleta de dados relacionados a poluição do ar em cidades. Para isto, foi desenvolvido um dispositivo através de um Arduino, equipado com sensores de CO e poeira e utilizando uma shield 3G/GRPS para o envio dos dados para o servidor, onde os dados coletados são armazenadas e disponibilizados através de um *Web Service* e um módulo GPS para adquirir a posição geográfica.

A proposta do *framework* assemelha-se bastante à proposta do trabalho de Devarakonda et al. (2013), por utilizar tanto a plataforma Arduino quanto por utilizar dispositivos para envio de dados e posicionamento GPS anexados ao Arduino.

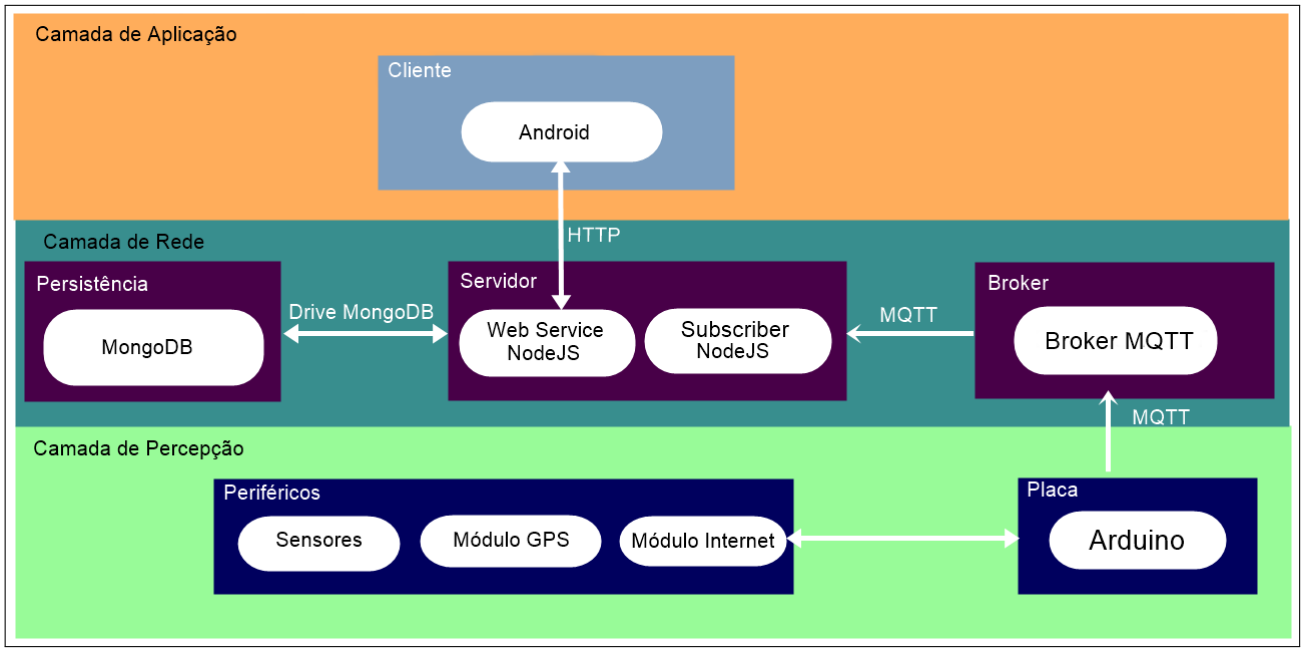
4 FRAMEWORK

Esta seção tem como objetivo apresentar o *framework* proposto no trabalho, apresentado na seção de **Introdução**.

4.1 Arquitetura do *Framework*

O *framework* proposto engloba desde o nível do Arduino até o nível do gerenciamento e distribuição dos dados coletados através de aplicações que cooperam entre si para garantir a funcionalidade esperada deste domínio. Com isso, o *framework* pode ser apresentado através de um arquitetura em camadas, apresentadas na figura a seguir.

Figura 8 – Arquitetura em Camadas do *Framework*



Fonte – Elaborada pelo autor

4.1.1 Camada de Percepção

Esta camada contém todo o conjunto de *hardware* necessário para que os dados ambientais possam ser coletados e enviados. Para isso, o *framework* utiliza a placa de prototipagem Arduino, que coleta as informações necessárias através de sensores específicos, incluindo a sua geolocalização através de um módulo GPS e envia estas informações através de um módulo Internet, utilizando o protocolo MQTT.

4.1.2 Camada de Rede

Esta camada é responsável pelo gerenciamento dos dados coletados na camada de percepção, em relação a persistência e a distribuição dos mesmos.

4.1.2.1 Broker

O *broker* é um intermediário entre o Servidor e o Arduino e tem como responsabilidade receber todas as coletas realizadas na camada de percepção e enviar esses dados para os seus *Subscribers* através do protocolo MQTT.

4.1.2.2 Servidor

O *Web Service* é responsável por disponibilizar as coletas realizadas e é configurável, para que o desenvolvedor possa definir como as coletas de dados realizadas deverão ser acessadas e também definir as consultas que podem ser realizadas sobre as mesmas. O *Subscriber* é uma aplicação responsável por receber e persistir todas as coletas realizadas na camada de percepção, através da comunicação com o *broker* utilizando o protocolo MQTT.

4.1.2.3 Persistência

É o local onde as coletas de dados ambientais realizadas são mantidas. Para isso, foi utilizado o SGBD orientado a documentos MongoDB.

4.1.3 Camada de aplicação

Esta camada é responsável por conter as aplicações clientes que irão consumir as coletas de dados armazenadas na base dados, através da comunicação com o *Web Service* desenvolvido utilizando o protocolo HTTP. As aplicações podem ser desenvolvidas para qualquer plataforma, como para o sistema operacional Android.

4.2 Utilização do *Framework*

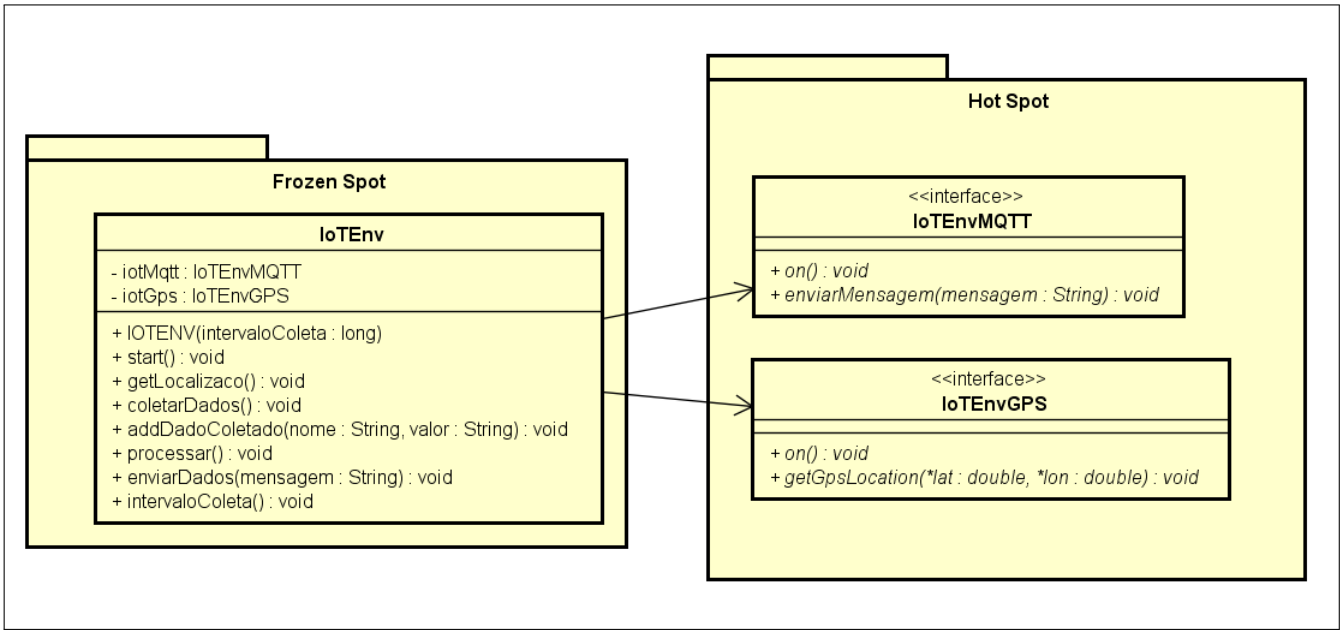
Para que uma aplicação de coleta de dados ambientais possa ser desenvolvida através do *framework*, o desenvolvedor deverá adaptá-lo para a aplicação que o mesmo deseja desenvolver, em relação ao conjunto de *hardware utilizado* e a forma como será realizada a

consulta dos dados. Será explicado a seguir como utilizar o *framework* de forma detalhada.

4.2.1 Arduino

O Arduino será utilizado como o dispositivo responsável por coletar as informações de determinados sensores de dados relacionados ao meio ambiente, sendo responsável também por gerenciar as mensagens que serão enviadas para o *broker* da aplicação contendo as informações da coleta e a sua posição geográfica. Para isso, o *framework* possui pontos flexíveis(*hot spots*) e imutáveis(*frozen spots*) no Arduino. Os *hot spots* são responsáveis pela comunicação com os dispositivos para comunicação com a Internet e localização GPS, e o *frozen spot* por coletar e estruturar os dados coletados a fim de enviá-los para o *broker* da aplicação. Esses pontos flexíveis e imutáveis são classes implementadas na linguagem C++ e deverão ser utilizadas pelo código Arduino implementado. Essas classes serão apresentadas a seguir.

Figura 9 – *Hot Spots e Frozen Spot* do Arduino



Fonte – Elaborada pelo autor

4.2.1.1 Hot Spots

As classes que representam os *hot spots* do *framework* são abstratas, ou seja, só é possível a instanciação de um objeto se estas classes tiverem seus métodos implementados. Portanto, estas classes devem ser implementadas de acordo com o *hardware* específico dos dispositivos responsáveis pela comunicação com a Internet e pela localização GPS, que em

sua maioria disponibilizam bibliotecas para facilitar o seu uso, que poderão ser utilizadas importando-as nessas classes. Além destas classes, também foram criadas outras duas de configuração, chamadas IoTEnvConfig, responsável por conter todas as configurações passíveis de alteração da aplicação e a classe IoTEnvData, responsável por conter o nome dos dados a serem coletados pela aplicação. Com isso, todos as informações contidas nestas classes ficarão disponíveis em todos os pontos da aplicação, fazendo com que as alterações que venham a surgir sejam feitas em um único local.

As classes que representam os *hot spots* do *framework* são:

4.2.1.1.1 IoTEnvMQTT

Esta classe abstrata é responsável pela comunicação com o *hardware* específico para conexão com a Internet utilizando o protocolo MQTT. Os seus métodos que devem ser implementados são:

- **on():** Este método é responsável por inicializar o *hardware* responsável pela comunicação com a Internet, deixando o dispositivo pronto para conexão e envio envio de dados.
- **enviarMensagem(String mensagem):** Este método é responsável pelo envio da mensagem recebida como parâmetro para o broker MQTT específico da aplicação. Haverá, em alguns casos, a necessidade de converter o formato da mensagem para o formato específico utilizado pelo *hardware* utilizado.

4.2.1.1.2 IoTEnvGPS

Esta classe abstrata é responsável pela comunicação com o *hardware* específico para localização GPS. Os seus métodos são:

- **on():** Este método é responsável por inicializar o *hardware* responsável pela localização GPS, deixando o dispositivo pronto para disponibilizar as suas coordenadas.
- **getGpsCoordinates(double *lat, double *lon):** Este método é responsável por adquirir as coordenadas do dispositivo. Este método recebe como parâmetro dois ponteiros do tipo *double*, que devem receber os valores das coordenadas adquiridas através do dispositivo.

4.2.1.2 Frozen Spot

4.2.1.2.1 IoTEnv

Esta classe representa o *frozen spot* do Arduino e já apresenta métodos implementados. Tem como finalidade disponibilizar uma estrutura simplificada para a coleta dos dados e é responsável pela comunicação com os *hot spots* do *framework*. Além disso, também é responsável por montar a mensagem no formato JSON que será enviada para o *broker* MQTT, cuja estrutura é a que se segue:

- Nome do dispositivo
- Localização (latitude e longitude)
- Dados Coletados

Figura 10 – Exemplo de uma mensagem gerada pela classe IoTEnv

```
{
  "dispositivo": "tcc-bruno",
  "localizacao": {
    "latitude": -4.9662261,
    "longitude": -39.0137485
  },
  "dados": [
    {
      "nome": "temperatura",
      "valor": "28"
    },
    {
      "nome": "umidade",
      "valor": "70"
    }
  ]
}
```

Fonte – Elaborada pelo autor

Esta mensagem é gerada graças ao auxílio de uma biblioteca chamada **ArduinoJson**[1], que facilita bastante a criação de mensagens no formato JSON em dispositivos Arduino. A classe IoTEnv apresenta os seguintes métodos:

[1] https://https://github.com/bblanchon/ArduinoJson

- **IOTENV(long intervaloColeta):** Este é o construtor da classe, que recebe como parâmetro o valor do intervalo que será utilizado entre uma coleta realizada e outra durante a execução da aplicação
- **start():** Este método é responsável por inicializar os *hot spots* do *framework*. É recomendado que esta função seja chamada dentro do método *setup()* do código arduino.
- **getLocalizacao():** Este método é responsável por adicionar a localização do dispositivo Arduino à mensagem que será enviada.
- **coletarDados():** Este método é responsável por conter toda a implementação necessária para que os dados possam ser coletados através dos sensores específicos da aplicação desenvolvida. Na prática, esta função não é previamente implementada, sendo necessário que o usuário implemente esta função no código Arduino.
- **addDadoColetado(String nome, String valor):** Esta função é responsável por adicionar à mensagem a ser enviada o valor do dado coletado pela aplicação através do uso de um sensor, ou seja, todos os dados obtidos a partir dos sensores utilizados no código Arduino deverão ser adicionados através deste método. Esta função recebe como parâmetro o nome do dado coletado através do uso de um sensor e o seu valor obtido, ambos do tipo String. A motivação para que o valor do dado coletado fosse do tipo String, é que os sensores podem retornar valores de vários tipos e a classe String do Arduino pode receber como parâmetro diversos tipos de valores. Logo, isso facilita bastante a inclusão de dados coletados à mensagem a ser gerada. Porém, é necessário tomar precauções posteriormente em uma possível conversão dos valores dos sensores no formato String, já que se feita de uma maneira inadequada, pode ocasionar em uma perca de precisão.
- **enviarDados():** Esta função é responsável por enviar a mensagem gerada para o broker MQTT determinado pela aplicação, ao se comunicar com a classe IoTMQTT.
- **intervaloColeta():** Esta função é responsável por aplicar um intervalo entre coleta de dados, após uma ter sido realizada e enviada.
- **processar():** Esta função é responsável por realizar o processo completo da

aplicação ao chamar todas as funções da classe, uma por vez. É indicado que esta função seja chamada dentro do método *loop()* do código Arduino, para que possa ser repetida continuamente.

4.2.2 Broker MQTT

O *Broker* MQTT é responsável por receber todas as mensagens provenientes do Arduino e enviar para todos os seus *Subscribers*. Existem vários *brokers* disponíveis, como por exemplo o *Mosquitto*[2], que é de fácil utilização e de código aberto.

Portanto, o desenvolvedor deverá configurar um *broker* MQTT para que as mensagens provenientes do Arduino possam ser recebidas.

4.2.3 Subscriber

A aplicação *Subscriber*[3] é responsável por receber todas as coletas realizadas provenientes do *broker* definido pelo desenvolvedor e persisti-las no banco de dados. Para isso, uma aplicação utilizando NodeJS foi disponibilizada para facilitar esta funcionalidade, sendo necessário apenas o desenvolvedor fornecer os dados de configuração da aplicação, em relação aos dados do *broker* e do banco de dados utilizado.

A aplicação espera receber mensagens no formato definido na aplicação Arduino, que representam uma coleta realizada. Após o recebimento da mensagem no formato correto, a aplicação anexa à coleta a data em que ela foi realizada, para que posteriormente consultas sobre possam ser realizadas em relação a intervalo de datas de coletas.

4.2.4 Web Service

O *Web Service*[4], como dito anteriormente, tem como objetivo disponibilizar todas as coletas de dados realizadas pelo *framework* através de um serviço RESTful. Foi desenvolvido utilizando NodeJS, que facilita muito o desenvolvimento de *Web Services RESTful* através do *framework Express JS*.

O *framework* busca facilitar o desenvolvimento de tais aplicações, por isso, o *Web Service* tem o intuito de ser configurável pelo desenvolvedor, para que a forma em que a consulta

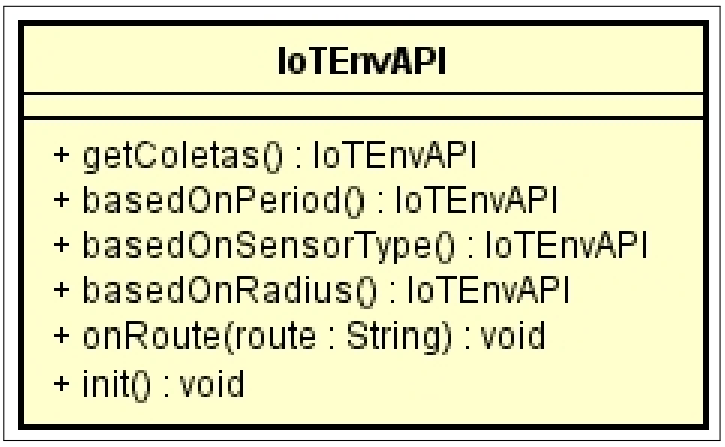
[2] https://mosquitto.org/
[3] https://github.com/brunobarretofreitas/IoT-Framework—Subscriber
[4] https://github.com/brunobarretofreitas/IoT-Framework—Web-Service

sobre as coletas realizadas sejam facilmente definidas. Para isso, um módulo foi desenvolvido com o intuito de definir filtros sobre as consultas, assim como os *endpoints* a serem acessados por aplicações clientes para consumir os dados.

Para que o Web Service fosse configurado facilmente, uma classe foi desenvolvida para facilitar tal propósito. Esta classe possui métodos para definir as consultas a serem realizadas sobre os dados. Esta classe se chama IoTEnvAPI e pode definir as consultas sobre as coletas realizadas baseadas em parâmetros como período de tempo, raio de distância e tipo de sensor. Os seus métodos são:

- **getColetas()**: Método responsável por retornar todas as coletas armazenadas no banco de dados.
- **basedOnPeriod()**: Método responsável por filtrar as coletadas baseadas em um período de tempo, com uma data inicial e uma data final
- **basedOnSensorType()**: Método responsável por filtrar as coletadas baseadas no tipo de sensor utilizado
- **basedOnRadius()**: Método responsável por filtrar as coletadas baseadas em um raio de distância a partir de uma determinada coordenada geográfica
- **onRoute()**: Método responsável por determinar o caminho a ser acessado para que uma determinada busca seja realizada.
- **init()**: Método responsável por executar o *Web Service*.

Figura 11 – Classe de definição do Web Service



Fonte – Elaborada pelo autor

O *Web Service* trabalha com o formato JSON, logo todas as coletas de dados serão disponibilizadas nesse formato.

4.2.5 Persistência

É responsável por manter todas as coletas de dados realizadas pelo Arduino. O desenvolvedor deverá utilizar o SGBD MongoDB, que é utilizado tanto pelo *Web Service* quanto pela aplicação Subscriber.

4.2.6 Cliente

O desenvolvedor deverá criar uma aplicação cliente que consiga se comunicar com o *Web Service* desenvolvido, de acordo com o que foi configurado no mesmo, em relação a como a consulta deverá ser realizada. Além disso, a aplicação deverá ser capaz de tratar mensagens no formato JSON, já que é o formato utilizado pelo *Web Service*.

4.3 Desenvolvimento de uma aplicação utilizando o *Framework* desenvolvido

Nesta etapa, foi desenvolvida uma aplicação de coleta de dados ambientais utilizando o *framework* proposto neste trabalho. A aplicação tem como objetivo coletar dados ambientais relacionados a poluição do ar, temperatura e umidade de cidades. A aplicação após ser desenvolvida, foi colocada em prática na cidade de Caucaia do Estado do Ceará. A seguir será descrito o processo de criação da aplicação em cada nível do *framework*.

4.3.1 Broker MQTT

O *broker* utilizado para que os dados coletados pela aplicação pudessem ser enviados foi o Mosquitto, que como já foi citado anteriormente, é de código aberto e de fácil utilização. O mesmo foi instalado em um serviço na nuvem para que pudesse ser acessado pelo Arduino.

4.3.2 Arduino

A versão utilizada na aplicação foi o Arduino MEGA, por fornecer um bom poder de processamento. Os periféricos utilizados, sensores e módulos responsáveis por localização GPS e comunicação Internet foram:

- Sensor de CO_2 MQ135
- Sensor de Temperatura e Umidade DHT11
- Módulo GSM/GPS SIMCOM 808

Para que o Arduino realizasse a coleta dos dados ambientais através dos periféricos citados, foi necessária a implementação das classes IoTEnvMQTT e IoTEnvGPS[5]. Na classe IoTEnvMQTT, foi implementado todo o código necessário para que o módulo GSM/GPS pudesse realizar conexão com a Internet e enviar dados via protocolo MQTT. Para isso, foi utilizada uma biblioteca chamada SIM800_MQTT[6], específica para o módulo em questão, promovendo todas as funcionalidades necessárias para conexão com a Internet.

Na classe IoTEnvGPS, foi implementado, assim como na IoTEnvMQTT, todo o código necessário para que o módulo GSM/GPS pudesse fornecer as coordenadas do dispositivo necessárias para a aplicação desenvolvida. Para isso, foi utilizada a biblioteca chamada GSMSHIELD[7], específica para o módulo em questão, na qual promove uma classe que fornece todas as funcionalidades necessárias para localização GPS.

Já no código Arduino, a classe IoTEnv foi importada e sua função **coletarDados()** foi implementada, onde todo o código necessário para a coleta dos dados através dos sensores foi implementado. Todos as configurações passíveis de serem modificadas, como os dados relacionados ao *broker* MQTT definido anteriormente, foram anexadas na classe IoTEnvConfig e o nome dos dados a serem coletados, CO_2, temperatura e umidade, foram anexados na classe IoTEnvData, para que pudessem ser acessados em qualquer lugar da aplicação. Para que o valor obtido através dos sensores fossem adicionados a mensagem a ser gerada pelo Arduino, foi utilizado o método **addDadoColetado(String nome, String valor)**, em que o nome do dado coletado de cada sensor utilizado e os seus valores obtidos são passados como parâmetro para esta função. O código Arduino desta aplicação será apresentado no Apêndice A deste trabalho.

Como um *broker* já havia sido implantado, foi possível testar se a aplicação Arduino estava funcionando corretamente, ao avaliar se os dados estavam sendo coletados e enviados para o *broker*. Para isso, foi utilizado um aplicativo chamado MQTT Client[8], que tem como funcionalidade ler todos os dados provenientes de um tópico específico de um broker MQTT. Com isso, observou-se que a mensagem gerada pela aplicação Arduino estava formatada de maneira correta e todas as informações estavam presentes.

[5] https://github.com/brunobarretofreitas/IoT-Framework—Arduino
[6] https://github.com/elementzonline/SIM800_MQTT
[7] https://github.com/jefflab/GSMSHIELD
[8] https://play.google.com/store/apps/details?id=com.deepeshc.mqttrec

Figura 12 – Mensagens recebidas no aplicativo MQTT Client

Fonte – Elaborada pelo autor

4.3.3 Subscriber

A aplicação *Subscriber*, como já dito anteriormente, necessita apenas dos dados em relação ao banco de dados da aplicação e do broker MQTT. Para isso, a aplicação disponibiliza um módulo NodeJS de configuração, em que deve ser fornecido o endereço, a porta e o tópico do *broker* e o endereço, a porta e a base de dados a ser utilizada.

Portanto, a aplicação *Subscriber* teve o seu módulo de configuração definido como mostra o código fonte a seguir e em seguida, foi colocada em um serviço na nuvem e executada, fazendo com que agora todas as coletas realizadas pelo Arduino, sejam persistidas no banco de dados.

Código-fonte 1 – Configuração da aplicação *Subscriber MQTT*

```
1
2 var config = {};
3
4 /*BROKER MQTT*/
5 config.mqttHost = "138.197.40.83";
6 config.mqttPort = 1883;
```

```
7  config.mqttTopico = "iotenv";
8
9  /*MongoDB*/
10 config.mongoHost = "localhost";
11 config.mongoPort = "27017";
12 config.mongoDatabase = "iotenv";
13
14 module.exports = config;
```

4.4 *Web Service*

O *Web Service* foi configurado para que as consultas de dados fossem realizadas baseadas em um raio de distância a partir de uma certa localização, em um período de tempo e no tipo do sensor utilizado. Como dito anteriormente, através da classe IoTEnvAPI, podemos definir as consultas facilmente, como mostra o código fonte a seguir.

Código-fonte 2 – Web Service para disponibilizar as coletas realizadas

```
1  var IoTEnvAPI = require( ./api/iotenvapi.js )();
2
3  IoTEnvAPI.getColetas().onRoute("/iotenv/coletas");
4  IoTEnvAPI.getColetas().basedOnPeriod("/iotenv/coletas/
      periodo");
5  IoTEnvAPI.getColetas().basedOnRadius().onRoute("/iotenv/
      raio");
6  IoTEnvAPI.getColetas().basedOnRadius().basedOnSensorType().
      onRoute("/iotenv/raio/dado");
7  IoTEnvAPI.getColetas().basedOnRadius().basedOnSensorType().
      basedOnPeriod().onRoute("/iotenv/raio/dado/periodo");
8
9  IoTEnvAPI.init();
```

Todos as consultas são feitas utilizando o método GET através das rotas definidas.

Após a configuração do *Web Service*, o mesmo foi colocado em um serviço na nuvem e executado, fazendo com que agora as coletas de dados armazenadas no banco de dados possam ser distribuídas para aplicações clientes.

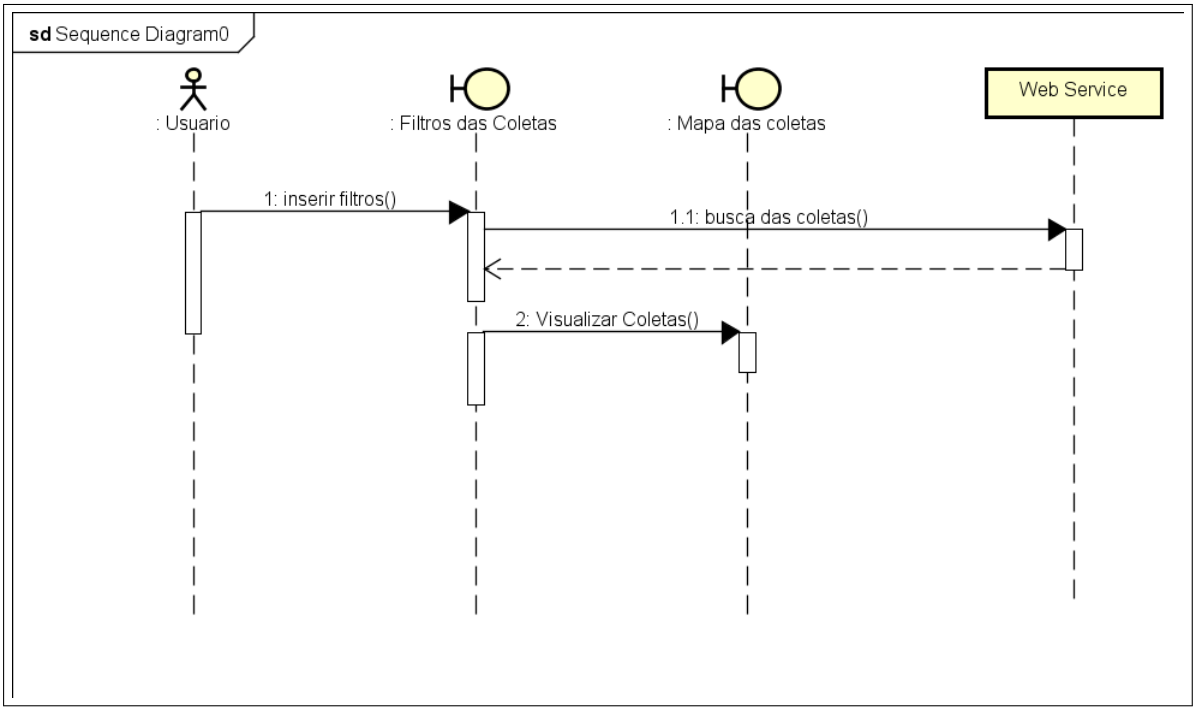
4.5 Cliente

Um aplicativo Android foi desenvolvido para consumir as coletas de dados realizadas. Para isso, o aplicativo realiza requisições para o *Web Service* configurado anteriormente, baseados nas consultas definidas, que permite o usuário da aplicação a escolher os parâmetros de consulta. Além disso, o aplicativo utiliza um mapa para que as coletas sejam visualizas em seus respectivos locais geográficos.

Figura 13 – Tela dos parâmetros de consultas das coletas de dados

Fonte – Elaborada pelo autor

Figura 14 – Processo de busca das coletas de dados ambientais do aplicativo



Fonte – Elaborada pelo autor

5 RESULTADOS E DISCUSSÃO

Nesta seção serão mostrados os resultados obtidos a partir da execução da aplicação de coleta de dados ambientais desenvolvida a partir do *framework* desenvolvido neste trabalho.

A aplicação foi colocada em execução em alguns locais da cidade de Caucaia - Ceará, nos dias 26 e 27 de Junho de 2017. O dispositivo Arduino realizou a coleta de dados ambientais por cerca de meia hora em cada local determinado.

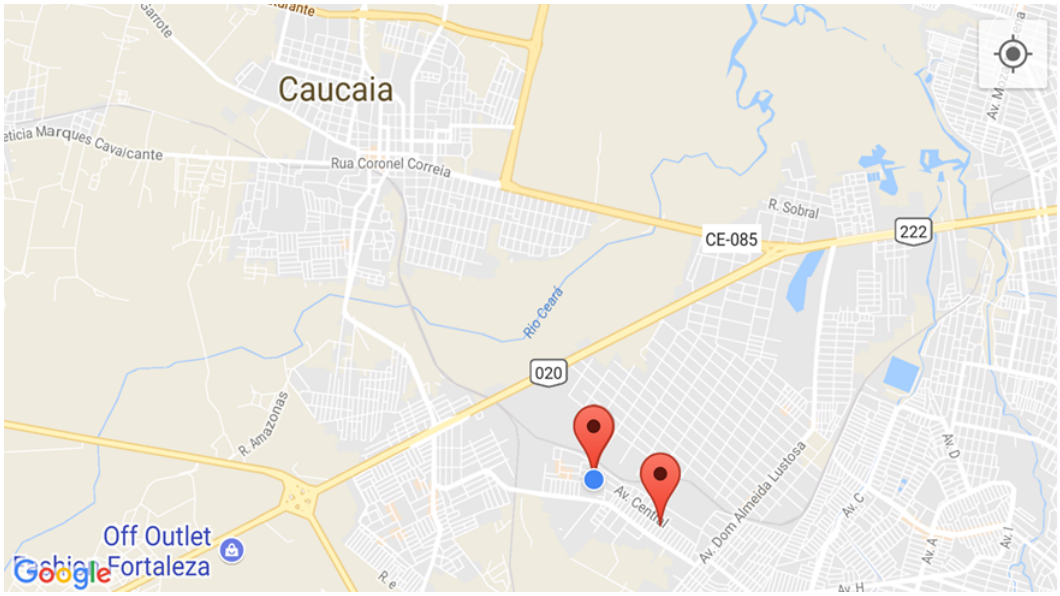
Na Figura 15, podemos ver os resultados gerados a partir da consulta configurada para filtrar os dados a partir de um período de tempo, no qual foi selecionado através do aplicativo desenvolvido apenas o dia 26 de Junho de 2017.

Figura 15 – Coletas realizadas no dia 26/06/2017

Fonte: Elaborada pelo autor

Na Figura 16, podemos ver os resultados gerados a partir da consulta configurada para filtrar os dados baseados em um raio de 3 Km a partir da localização do usuário da aplicação Android.

Figura 16 – Coletas realizadas em um raio de 3 Km a partir da localização do usuário



Fonte: Elaborada pelo autor

Na Figura 17, podemos ver os resultados gerados a partir da consulta configurada para filtrar os dados baseados no tipo de sensor utilizado e em um Raio de distância a partir da localização do usuário, no qual foi solicitado o sensor de CO_2 e um raio de 20 Km.

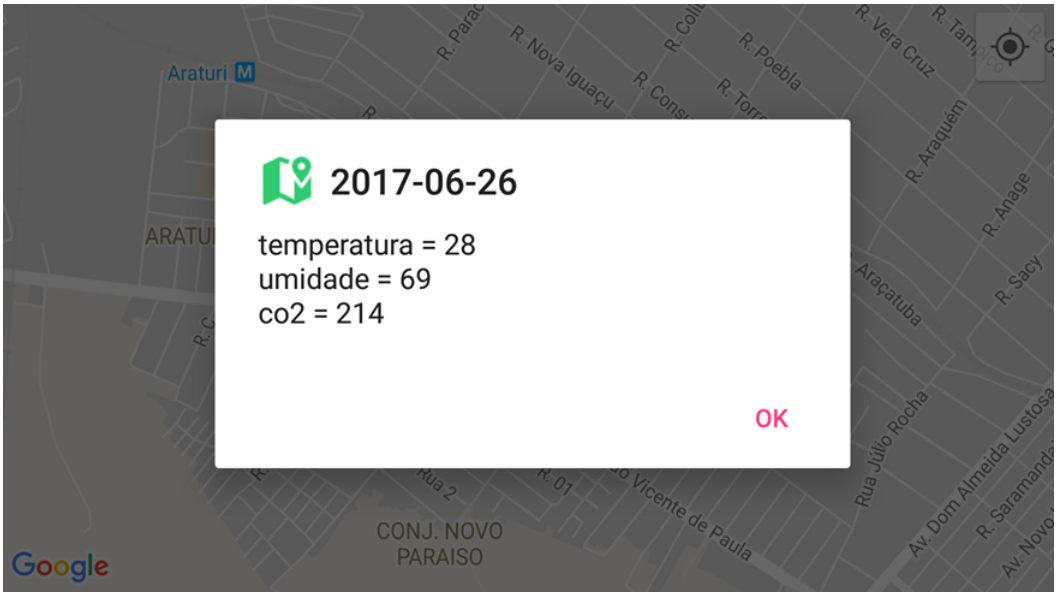
Figura 17 – Coletas realizadas que contenham o sensor de CO_2 em um Raio de 20 Km

Fonte: Elaborada pelo autor

Na Figura 18, podemos ver as informações encontradas em uma coleta realizada, ao

clicar em um marcador presente no mapa.

Figura 18 – Informações de uma coleta



Fonte: Elaborada pelo autor

5.1 Problemas

Durante a implementação da aplicação a partir do *framework* desenvolvido, alguns problemas foram encontrado, como na implementação das classes do *framework* responsáveis pelos módulos de Internet e GPS do Arduino para a aplicação desenvolvida. Primeiramente, o módulo SIM808 apresentou um desempenho ruim, levando bastante tempo para que a localização GPS fosse adquirida e muitas vezes foi necessário reiniciar o dispositivo por apresentar erros constantes. Segundo, o módulo também apresentou bastante dificuldade para realizar a comunicação utilizando o protocolo MQTT, levando muito tempo para que as informações pudessem ser enviadas para o *broker*.

6 CONSIDERAÇÕES FINAIS

Este trabalho desenvolveu um *framework* para coletas de dados ambientais em cidades inteligentes, uma vez que foi identificado trabalhos que possuem grandes semelhanças nos procedimentos realizados para alcançar este fim.

O *Framework* desenvolvido disponibiliza softwares adaptáveis que auxiliam as etapas necessárias para que o processo de coletar dados ambientais em cidades inteligentes seja realizado. As etapas cobrem desde a programação do Arduino junto com sensores e módulos GPS e Internet para que as coletas de dados sejam realizas, até a disponibilização destas coletas através de um *Web Service RESTful* configurável.

Foi realizado ainda o desenvolvimento de uma aplicação a partir do *framework* desenvolvido, para que coletas de dados ambientais em relação à CO_2, temperatura e umidade fossem realizadas. Após todo o processo, desde a implementação do Arduino até a configuração do *Web Service*, uma aplicação Android foi desenvolvida para que as coletas realizadas pudessem ser visualizadas em um mapa. Com isso, a aplicação foi colocada em prática na cidade de Caucaia-CE nos dias 26/06/2017 e 27/06/2017.

Como trabalhos futuros, seria interessante que o *Web Service* pudesse ser configurável para não só apenas filtrar as coletas realizadas com base em período de tempo, raio de distância e tipo de sensor, mas que fosse possível definir de maneira simples qualquer tipo de busca genérica sobre as coletas realizadas. Além disso, seria de bastante valor um estudo sobre o *framework* desenvolvido em relação a sua capacidade de facilitar o desenvolvimento de aplicações de coleta de dados ambientais.

REFERÊNCIAS

ANDROID. **Arduino - História**. 2016. Acessado em 05 de Julho 2016. Disponível em: <https://www.android.com/history>.

ANDROID SOURCE. **Arquitetura do Android [manual online]**. [S.l.], 2016. Android Source. Disponível em: <http://source.android.com/index.html>. Acesso em: 10 abr. 2016.

ARDUINO. **Arduino - Introduction**. 2016. Acessado em 05 de Julho 2016. Disponível em: <https://www.arduino.cc>.

AREVALO, G. B. **Architectural Description of Object Oriented Frameworks**. Tese (Doutorado) — Master Thesis. Ecole des Mines de Nantes and Vrije Universiteit Brussels, 2000.

ASHTON, K. That 'internet of things' thing. **RFiD Journal**, v. 22, n. 7, p. 97–114, 2009.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. **Computer networks**, Elsevier, v. 54, n. 15, p. 2787–2805, 2010.

CROCKFORD, D. The application/json media type for javascript object notation (json). 2006.

DEVARAKONDA, S.; SEVUSU, P.; LIU, H.; LIU, R.; IFTODE, L.; NATH, B. Real-time air quality monitoring through mobile sensing in metropolitan areas. In: ACM. **Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing**. [S.l.], 2013. p. 15.

DEVELOPERS, A. **What is android**. [S.l.]: Android Developers, http://developer. android. com/guide/basics/what-is-android. html, accessed May, 2011.

GUBBI, J.; BUYYA, R.; MARUSIC, S.; PALANISWAMI, M. Internet of things (iot): A vision, architectural elements, and future directions. **Future Generation Computer Systems**, Elsevier, v. 29, n. 7, p. 1645–1660, 2013.

HUNKELER, U.; TRUONG, H. L.; STANFORD-CLARK, A. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In: IEEE. **Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on**. [S.l.], 2008. p. 791–798.

JIN, J.; GUBBI, J.; MARUSIC, S.; PALANISWAMI, M. An information framework for creating a smart city through internet of things. **Internet of Things Journal, IEEE**, IEEE, v. 1, n. 2, p. 112–121, 2014.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. **Journal of object-oriented programming**, v. 1, n. 2, p. 22–35, 1988.

LECHETA, R. R. **Google Android-3ª Edição: Aprenda a criar aplicações para dispositivos móveis com o Android SDK**. [S.l.]: Novatec Editora, 2013.

MARKIEWICZ, M. E.; LUCENA, C. J. de. Object oriented framework development. **Crossroads**, ACM, v. 7, n. 4, p. 3–9, 2001.

MCROBERTS, M. **Arduino Básico, 2ª Edição**. [S.l.]: Novatec, 2015. v. 2.

MENDEZ, D.; PEREZ, A. J.; LABRADOR, M. A.; MARRON, J. J. P-sense: A participatory sensing system for air pollution monitoring and control. In: IEEE. **Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on**. [S.l.], 2011. p. 344–347.

MENG, J.; MEI, S.; YAN, Z. Restful web services: A solution for distributed data integration. In: IEEE. **Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on**. [S.l.], 2009. p. 1–4.

MIORANDI, D.; SICARI, S.; PELLEGRINI, F. D.; CHLAMTAC, I. Internet of things: Vision, applications and research challenges. **Ad Hoc Networks**, Elsevier, v. 10, n. 7, p. 1497–1516, 2012.

MQTT. **Frequently Asked Questions**. 2017. Acessado em 04 de Julho 2017. Disponível em: <https://mqtt.org/faq>.

STARTUPBOOTCAMP. **Cidade Inteligente [manual online]**. [S.l.], 2016. Portal Corporativo. Disponível em: <http://www.startupbootcamp.org/assets/images/blog/smart-city-living/infographic.png>. Acesso em: 17 maio. 2016.

SU, K.; LI, J.; FU, H. Smart city and the applications. In: IEEE. **Electronics, Communications and Control (ICECC), 2011 International Conference on**. [S.l.], 2011. p. 1028–1031.

VAGNOLI, C.; MARTELLI, F.; FILIPPIS, T. D.; LONARDO, S. D.; GIOLI, B.; GUALTIERI, G.; MATESE, A.; ROCCHI, L.; TOSCANO, P.; ZALDEI, A. The sensorwebbike for air quality monitoring in a smart city. In: IET. **Future Intelligent Cities, IET Conference on**. [S.l.], 2014. p. 1–4.

WOLFGANG, P. **Design patterns for object-oriented software development**. [S.l.]: Reading, Mass.: Addison-Wesley, 1994.

ZANELLA, A.; BUI, N.; CASTELLANI, A.; VANGELISTA, L.; ZORZI, M. Internet of things for smart cities. **Internet of Things Journal, IEEE**, IEEE, v. 1, n. 1, p. 22–32, 2014.

APÊNDICE A – CÓDIGO ARDUINO DA APLICAÇÃO DESENVOLVIDA

Código-fonte 3 – Código Arduino da aplicação desenvolvida

```
#include "IoTEnv.h"
#include <SimpleDHT.h>
#define  pinDHT11 A1

IOTENV iot(30); int MQ135 = A0; SimpleDHT11 dht11;

void IOTENV::coletarDados(){
    int co2 = analogRead(MQ135);
    byte temperature = 0;
    byte humidity = 0;
    if (dht11.read(pinDHT11, &temperature, &humidity, NULL)
       ) {
      Serial.println("Read DHT11 failed.");
      return;
    }
  iot.addIndice(_CO2, String(co2));
  iot.addIndice(_TEMPERATURA, String(temperatura));
  iot.addIndice(_UMIDADE, String(umidade));
}

void setup(){
  Serial.begin(9600);
  iot.start();
}

void loop(){
  iot.processar();
}
```

APÊNDICE B – IMPLEMENTAÇÃO DA CLASSE ABSTRATA IOTENVMQTT DA APLICAÇÃO DESENVOLVIDA

Código-fonte 4 – Classe IoTEnvMQTT implementada para o módulo GSM/GPS SIM808

```
1    #include "IoTEnvMQTT.h"
2  #include "GSM_MQTT.h"
3
4  String MQTT_HOST = _broker_url, MQTT_PORT = _broker_porta,
       MQTT_TOPICO = _broker_topico;
5  GSM_MQTT MQTT(20);
6
7  void IOTENVMQTT::on(){MQTT.gsmOn();}
8
9  void IOTENVMQTT::enviarMensagem(String mensagem){
10   bool mensagem_enviada = false;
11   MQTT.begin();
12   delay(5000);
13   while(!mensagem_enviada){
14     if(MQTT.available()){
15       Serial.println("MQTT Dispon vel");
16       char m[mensagem.length()];
17       mensagem.toCharArray(m, mensagem.length());
18       MQTT.publish(0, 0, 0, "iotmessage", topi, m);
19     }
20     mensagem_enviada = MQTT.publishSent();
21     Serial.print("O MQTTSENT DEU ");
22     Serial.println(mensagem_enviada);
23     MQTT.processing();
24     delay(5000);
25   }
26 }
```

APÊNDICE C – IMPLEMENTAÇÃO DA CLASSE ABSTRATA IOTENVGPS DA APLICAÇÃO DESENVOLVIDA

Código-fonte 5 – Classe IoTEnvGPS implementada para o módulo GSM/GPS SIM808

```
#include "IoTEnvGPS.h"
#include "gps.h"
#include <stdlib.h>
GPSGSM gps_gsm;

void IOTENVGPS::on(){
    if (gps_gsm.attachGPS()){
  Serial.println("status=GPSREADY");
  delay(20000); //Time for fixing
    }else{Serial.println("status=ERROR");}
}

void IOTENVGPS::getGPSLocation(double *lat, double *lon){
    int stat = 0;
  while(stat!=2 && stat!=3){
      stat=gps_gsm.getStat();
      switch(stat){
        case 1: Serial.println("NOT FIXED"); break;
        case 2: Serial.println("2D FIXED"); break;
        case 3: Serial.println("3D FIXED"); break;
      }
    }
  char lo[15],la[15],alt[15],time[20],vel[15] ;
  gps_gsm.getPar(la,lo,alt,time,vel);
  *lat = atof(la);
  *lon = atof(lo);
}
```