

João Higo Sousa Nunes  
João Vitor Miranda Roma

# **Aplicação da arquitetura SOA para a integração de sistemas embarcados de baixo custo**

São Luís - MA

2017

João Higo Sousa Nunes  
João Vitor Miranda Roma

## **Aplicação da arquitetura SOA para a integração de sistemas embarcados de baixo custo**

Trabalho apresentado ao Curso Bacharelado Interdisciplinar em Ciência e Tecnologia, da Universidade Federal do Maranhão, como pré-requisito para a elaboração do Trabalho de Contextualização e Integralização Curricular.

Universidade Federal do Maranhão  
Bacharelado Interdisciplinar em Ciência e Tecnologia

Orientador: Prof. Dr. Inaldo Capistrano Costa  
Coorientador: Prof. Me. Luis Claudio de Oliveira Silva

São Luís - MA

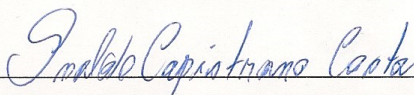
2017

João Higo Sousa Nunes  
João Vitor Miranda Roma

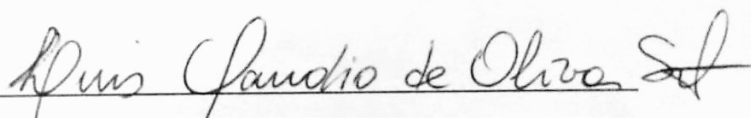
## **Aplicação da arquitetura SOA para a integração de sistemas embarcados de baixo custo**

Trabalho apresentado ao Curso Bacharelado Interdisciplinar em Ciência e Tecnologia, da Universidade Federal do Maranhão, como pré-requisito para a elaboração do Trabalho de Contextualização e Integralização Curricular.

Trabalho aprovado. São Luís - MA, 27 de Janeiro de 2017:



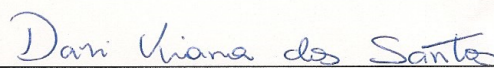
Prof. Dr. Inaldo Capistrano Costa  
Orientador



Prof. Me. Luis Claudio de O. Silva  
Orientador



Prof. Dr. Sérgio Souza Costa  
Universidade Federal do Maranhão



Prof. Dr. Davi Viana dos Santos  
Universidade Federal do Maranhão

São Luís - MA  
2017

# Resumo

A Internet das Coisas (em inglês, IoT) é um paradigma tecnológico que tem como objetivo conectar dispositivos eletrônicos entre si e com a Internet. Para essa conexão, é ressaltado o uso de sensores e atuadores em conjunto com sistemas embarcados microcontroladores conectados em uma rede. Diversas empresas vêm adotando o protocolo REST pelas vantagens de escalabilidade e mais leve, além da tendência estar apontando para um menor uso de dados e protocolos para dispositivos mais básicos, simplificando o estabelecimento da comunicação. Este trabalho se propõe a construir um sistema de baixo custo para Internet das Coisas, por meio da Arquitetura Orientada a Serviços. Para alcançar esse objetivo, neste trabalho foi construído um protótipo que serve como prova de conceito para o uso da REST como tecnologia integradora entre várias plataformas, demonstrando sua capacidade de expansão para diversas aplicações.

**Palavras-chave:** Internet das Coisas. SOA. REST. NodeMCU. Android.

# Abstract

The Internet of Things (IoT) is a technological paradigm that aims to connect electronic devices to each other and to the Internet. For this connection, it is emphasized the use of sensors and actuators in conjunction with embedded systems microcontrollers connected in a network. Several companies have been adopting the REST protocol for the advantages of scalability and lighter, besides the trend is pointing to a smaller use of data and protocols for more basic devices, simplifying the establishment of communication. This work proposes to build a low cost system for the Internet of Things, through the Service Oriented Architecture. In order to achieve this goal, a prototype was built to serve as a proof of concept for the use of REST as an integrating technology among several platforms, demonstrating its capacity to expand to several applications.

**Keywords:** Internet of Things. SOA. REST. NodeMCU. Android.

# Lista de ilustrações

Figura 1 – Arquitetura básica SOA . . . . .	15
Figura 2 – Visão geral da arquitetura . . . . .	26
Figura 3 – Classes dos serviços . . . . .	29
Figura 4 – Tabelas do banco de dados . . . . .	30
Figura 5 – Convertendo dados para JSON . . . . .	30
Figura 6 – Tratamento de erros com dados recebidos . . . . .	31
Figura 7 – Requisição GET . . . . .	32
Figura 8 – Esquema elétrico . . . . .	32
Figura 9 – Comandos prévios . . . . .	33
Figura 10 – Função "setup" . . . . .	34
Figura 11 – Layout XML do aplicativo . . . . .	35
Figura 12 – Montagem na Protoboard . . . . .	36
Figura 13 – Serial . . . . .	37
Figura 14 – Recebimento dos dados pelo servidor . . . . .	38
Figura 15 – Informações armazenadas do banco de dados . . . . .	38
Figura 16 – Interface do aplicativo . . . . .	39
Figura 17 – Tela de criação de projeto . . . . .	45
Figura 18 – Estrutura do projeto . . . . .	46
Figura 19 – Classes dos serviços . . . . .	46
Figura 20 – Caminho do serviço . . . . .	47
Figura 21 – Requisição . . . . .	47
Figura 22 – Criando banco de dados . . . . .	48
Figura 23 – Definindo <i>login</i> do banco de dados . . . . .	49
Figura 24 – Executar SQL . . . . .	49
Figura 25 – Tabelas do banco de dados . . . . .	50
Figura 26 – Adicionando driver . . . . .	51
Figura 27 – Classes de conexão . . . . .	51
Figura 28 – Classe DAO . . . . .	52
Figura 29 – Classe GerenciarDAO . . . . .	53
Figura 30 – Classe GerenciarDAO . . . . .	54
Figura 31 – Adicionando biblioteca gson . . . . .	54
Figura 32 – Convertendo JSON para objeto Java . . . . .	55
Figura 33 – Método GET . . . . .	55
Figura 34 – Método GET . . . . .	56
Figura 35 – Requisição GET . . . . .	57
Figura 36 – Requisição POST . . . . .	57

Figura 37 – NodeMCU . . . . .	58
Figura 38 – Sensor de Movimento . . . . .	59
Figura 39 – Sensor de Temperatura . . . . .	60
Figura 40 – Esquema elétrico . . . . .	61
Figura 41 – Menu Preferências . . . . .	62
Figura 42 – Preferências . . . . .	63
Figura 43 – Gerenciar Placas . . . . .	64
Figura 44 – Instalar biblioteca . . . . .	64
Figura 45 – Escolhendo a placa NodeMCU . . . . .	65
Figura 46 – Escolha do exemplo . . . . .	66
Figura 47 – Comandos prévios . . . . .	67
Figura 48 – Diagrama de pinos . . . . .	67
Figura 49 – Aplicativo Executar . . . . .	68
Figura 50 – Executar comando cmd . . . . .	69
Figura 51 – Comando ipconfig no Prompt . . . . .	69
Figura 52 – Função <i>setup</i> . . . . .	70
Figura 53 – Layout XML do aplicativo . . . . .	75
Figura 54 – Permissões de acesso . . . . .	75
Figura 55 – Classe de conexão . . . . .	76
Figura 56 – Conversão do JSON . . . . .	77
Figura 57 – Classe principal . . . . .	78

# Lista de tabelas

Tabela 1 – Métodos do REST . . . . .	18
Tabela 2 – Resumo da análise de tecnologias para plataformas embarcadas e móveis	23
Tabela 3 – Resumo da análise de tecnologias para servidor . . . . .	23
Tabela 4 – Preços dos componentes . . . . .	36



# Lista de abreviaturas e siglas

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACM	Association for Computing Machinery
API	Application Programming Interface
BD	Banco de Dados
CORBA	Common Object Request Broker Architecture
CoAP	Constrained Application Protocol
CoRE	Constrained RESTful Environment
EJB	Enterprise JavaBeans
EXI	Efficient XML Interchange
FPGA	Field Programmable Gate Array
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IoT	Internet of Things
JDBC	Java Database Connectivity
JDK	Java Development Kit
JPA	Java Persistence API
JSON	JavaScript Object Notation
LAN	Local Area Network

LED	Light Emitting Diode
MCU	Microcontroller Unit
OPC-UA	Open Platform Communications Unified Architecture
PDA	Personal Digital Assistant
REST	Representational State Transfer
RFID	Radio-Frequency Identification
SOAP	Simple Object Access Protocol
SOA	Service-Oriented Architecture
SOCRADES	Service-Oriented Cross-layer infrastructure for Distributed smart Embedded devices
SQL	Structured Query Language
SSID	Service Set Identifier
TCP	Transmission Control Protocol
TI	Tecnologia da Informação
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
WSDL	Web Services Description Language
WS	Web Service
WoT	Web of Things
XML	Extensible Markup Language

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>12</b>
<b>1.1</b>	<b>Objetivos</b>	<b>12</b>
1.1.1	Objetivo Geral	12
1.1.2	Objetivo Específico	13
<b>1.2</b>	<b>Organização do Documento</b>	<b>13</b>
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA</b>	<b>14</b>
<b>2.1</b>	<b>Internet das Coisas</b>	<b>14</b>
<b>2.2</b>	<b>Arquitetura Orientada a Serviços</b>	<b>15</b>
2.2.1	Serviços web baseados em SOAP	16
2.2.2	Serviços web baseados em REST	16
<b>2.3</b>	<b>Trabalhos Relacionados</b>	<b>18</b>
2.3.1	Estudo Comparativo de Tecnologias	18
2.3.2	Tabelas comparativas	22
2.3.3	Análise do Estudo Comparativo	24
<b>3</b>	<b>METODOLOGIA</b>	<b>25</b>
<b>3.1</b>	<b>Servidor</b>	<b>26</b>
<b>3.2</b>	<b>Cliente Embarcado</b>	<b>27</b>
<b>3.3</b>	<b>Cliente Móvel</b>	<b>27</b>
<b>4</b>	<b>PROVA DE CONCEITO</b>	<b>28</b>
<b>4.1</b>	<b>Servidor</b>	<b>28</b>
4.1.1	Banco de dados	29
4.1.2	Trabalhando e prevenindo erros com JSON	30
4.1.3	Funcionamento do acesso	31
<b>4.2</b>	<b>Cliente Embarcado</b>	<b>32</b>
<b>4.3</b>	<b>Cliente Móvel</b>	<b>35</b>
<b>4.4</b>	<b>Resultados</b>	<b>36</b>
<b>5</b>	<b>CONSIDERAÇÕES FINAIS</b>	<b>40</b>
	<b>REFERÊNCIAS</b>	<b>42</b>
	<b>APÊNDICE A – CRIAÇÃO DE SERVIDOR REST COM BANCO DE DADOS</b>	<b>45</b>

A.1	Criando um Servidor . . . . .	45
A.2	Criando banco de dados Derby . . . . .	47
A.3	Conectando o Servidor ao banco de dados . . . . .	50
A.4	Trabalhando com JSON . . . . .	54
A.5	Funcionamento dos serviços . . . . .	55
<p><b>APÊNDICE B – PROCEDIMENTOS PARA CONEXÃO DO MÓ- DULO NODEMCU A UM SERVIDOR REST . . .</b></p>		<b>58</b>
B.1	Código do Programa . . . . .	71
<p><b>APÊNDICE C – CRIANDO APLICAÇÃO CONSUMIDOR DO SER- VIÇO NO ANDROID . . . . .</b></p>		<b>74</b>
C.1	Definindo o layout de apresentação e funcionalidades . . . . .	74
C.2	Conectando o aplicativo ao servidor . . . . .	75
C.3	Trabalhando com JSON . . . . .	76
C.4	Chamando os recursos . . . . .	77

# 1 Introdução

O ser humano sempre buscou meios que facilitassem suas tarefas cotidianas. A Internet das Coisas (em inglês, *Internet of Things*, ou IoT) é um paradigma tecnológico que tem como objetivo conectar dispositivos eletrônicos entre si e com a Internet. Permitindo a coleta e a troca de dados entre eles, dessa forma, criando maior integração entre os sistemas e o mundo real, em busca de melhorias na eficiência dos serviços oferecidos ([ATZORI; IERA; MORABITO, 2010](#)) ([MATTERN; FLOERKEMEIER, 2010](#)).

Segundo [Friess \(2013\)](#), a IoT pode ser aplicada a casas, prédios, indústrias, transportes, hospitais, sistemas de gerenciamento de água, energia, e alimentos, entre outros ambientes, tornando-os inteligentes, ou seja, utilizando a computação de forma imperceptível para melhorar suas atividades. Para o desenvolvimento de um sistema IoT, é ressaltado o uso de sensores e atuadores conectados em uma rede, coletando e enviando informações. Isso pode ser usado para monitorar e avaliar características de um ambiente, como a temperatura e os horários com maior ou menor frequência de movimento neste ambiente.

Para [Suryadevara e Mukhopadhyay \(2012\)](#), em condições normais, uma pessoa executa atividades diárias em intervalos regulares de tempo. Isso implica que a pessoa está mentalmente e fisicamente apta e levando uma vida regular, isso nos diz que o bem-estar geral da pessoa está em um certo padrão. Segundo os autores, se houver declínio ou mudança na atividade regular, então o bem-estar da pessoa não está no estado normal. O bem-estar de idosos pode ser avaliado para prever situações inseguras durante o monitoramento de atividades regulares, sendo possível fornecer ajuda antes que uma situação imprevista aconteça.

## 1.1 Objetivos

### 1.1.1 Objetivo Geral

No presente trabalho, buscou-se implementar um sistema IoT com Orientação a Serviços. Tal sistema deve seguir protocolos pré-selecionados através de análise de outros trabalhos, que indicam as tecnologias que mais se adequam a ele. O sistema deve fazer a comunicação entre plataformas heterogêneas, ou seja, com linguagens de programações diferentes, através de uma arquitetura orientada a serviços.

A abordagem se baseia na arquitetura pretendida, um servidor, um cliente embarcado e um cliente móvel, para isso, a tecnologia apontada deve ser suportada por todos os dispositivos e não deve requerer muitos recursos dos tais, pois pretende-se que o sistema

seja de baixo custo, o que implica em pouco poder de processamento. Então, após todas as tecnologias definidas, o trabalho passará por uma prova de conceitos aplicada, onde será montado um exemplo prático que se utiliza da tecnologia em todos os componentes citados.

### 1.1.2 Objetivo Específico

Será feito como prova de conceito um sistema de monitoramento domiciliar baseado em uma rede de sensores que enviam informações para um servidor. A prova de conceito é formada por um servidor, executado em um computador pessoal, uma plataforma embarcada microcontroladora, conectada aos sensores de temperatura e movimento, e uma aplicação móvel, um *smartphone* com um aplicativo desenvolvido para interagir com o sistema, recebendo as informações mais recentes dos sensores, quando requisitado.

Dentre as diversas tecnologias existentes para integração da ideia de IoT, este trabalho se propõe a construir um protótipo que sirva como prova de conceito para o uso do protocolo REST (*Representational State Transfer*) como tecnologia integradora entre vários dispositivos, várias camadas (servidor e clientes) e várias tecnologias (tanto móveis quanto embarcadas).

## 1.2 Organização do Documento

- Capítulo 2 - Fundamentação Teórica: Neste capítulo, são mostrados os conceitos de Internet das Coisas e Arquitetura Orientada a Serviços, detalhando seus protocolos, conceitos cujo o entendimento é essencial para o desenvolvimento deste projeto. Também é apresentado o resultado de uma busca de artigos nos quais são discutidas tecnologias que são utilizadas em aplicações voltadas à IoT.
- Capítulo 3 - Metodologia: Neste capítulo, é apresentada a metodologia utilizada no trabalho, detalhes da arquitetura proposta, e escolha das tecnologias a serem utilizadas no experimento.
- Capítulo 4 - Prova de Conceito: Neste capítulo, é descrita em detalhes a implementação do projeto. Os resultados da prova de conceito são apresentados e discutidos.
- Capítulo 5 - Considerações Finais: No final do trabalho, são constatadas as descobertas e conclusões obtidas com a análise dos resultados dos testes feitos na prova de conceito.
- Apêndices: Nos apêndices deste documento, são encontrados tutoriais detalhados de como reproduzir todo o experimento proposto neste trabalho.

## 2 Fundamentação Teórica

### 2.1 Internet das Coisas

Um dispositivo, no que diz respeito à Internet das Coisas, é um equipamento com capacidades de comunicação, que possui sensores e atuadores, e é capaz de coletar, armazenar e processar dados. É um objeto do mundo físico ou virtual que é capaz de ser identificado e integrado em redes de comunicação, pode ser genericamente denominado de Coisa ([RECOMMENDATION, 2012](#)).

De acordo com Cisco Internet Business Solutions Group, em 2003, havia uma proporção de 0,08 dispositivos conectados à Internet para cada pessoa no mundo. Em 2010, após o crescimento de smartphones e tablets no mercado e a miniaturização de dispositivos eletrônicos, esse número aumentou para 1,84. Estima-se que o número de dispositivos conectados à Internet ultrapassou o número de pessoas na Terra por volta do ano de 2008 ([EVANS, 2011](#)).

Os computadores, e conseqüentemente a Internet, são normalmente dependentes de seres humanos para conseguirem informações. Praticamente todos os dados que existem na Internet foram criados por humanos, seja digitando, gravando, tirando fotos, escaneando, ou qualquer outra forma de captura de dados. O problema é que as pessoas têm tempo, atenção e precisão limitados, ou seja, não somos muito bons em gerar dados. Para [Ashton \(2009\)](#),

"Se tivéssemos computadores que soubessem tudo o que havia para saber sobre as coisas - utilizando dados que elas mesmo reuniriam de forma independente - seríamos capazes de monitorar e contar tudo, e reduzir muito o desperdício, perdas e custos. Nós saberíamos quando as coisas precisariam de substituição, reparo ou troca, e se elas estariam novas ou se já passaram da sua vida útil."([ASHTON, 2009](#))

Como exemplo disso, já existem alguns produtos IoT no mercado, como o *Nest Learning Thermostat* <sup>1</sup>, um termostato inteligente que aprende as preferências de temperatura do usuário e se ajusta automaticamente nos horários ideais, obtendo uma economia de 10 a 15% no consumo de energia. Exemplos como este mostram que a pesquisa acadêmica sobre Internet das Coisas está indo ao encontro de uma necessidade de mercado.

O desenvolvimento atual da IoT tem sido liderado por empresas proprietárias, o que tem como conseqüências o atraso no desenvolvimento e o encarecimento do produto final. Para atingirmos uma independência e desenvolvermos um sistema IoT, é ressaltado o uso de

<sup>1</sup> Internet Of Things Wiki. < <http://internetofthingswiki.com/nest-learning-thermostat/559/>>.

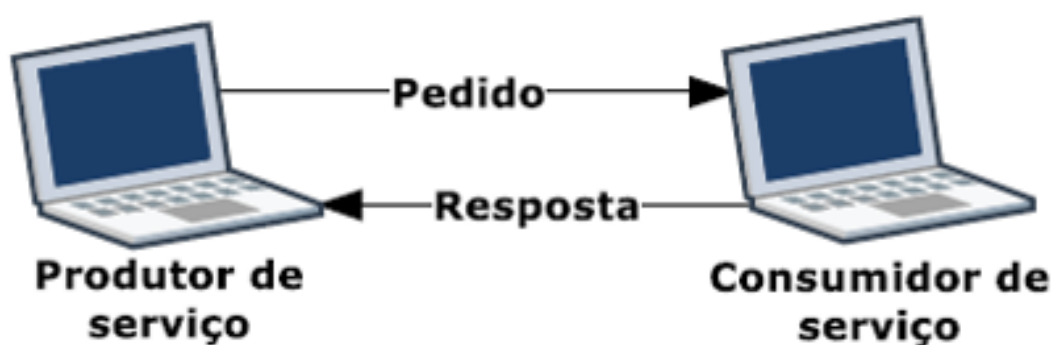
sensores e atuadores em conjunto com sistemas embarcados microcontroladores conectados em uma rede. A conectividade e interação destes dispositivos são primordiais para o bom funcionamento dos serviços ofertados por esta revolução tecnológica. Mas a busca por uma interação entre dispositivos variados acarreta em problemas de compatibilidade entre arquiteturas, sistemas operacionais, linguagens de programação, protocolos de comunicação, etc.

## 2.2 Arquitetura Orientada a Serviços

Uma Arquitetura Orientada a Serviços é um estilo de arquitetura cujo objetivo é conectar agentes de software por meio de serviços. Um serviço é uma unidade de trabalho feita por um provedor de serviços para alcançar os resultados finais desejados para um consumidor de serviços. Ambos provedor e consumidor são papéis desempenhados por agentes de software em nome de seus proprietários (HE, 2003).

A arquitetura básica consiste em um produtor e um consumidor de serviços. O consumidor manda uma mensagem com a requisição ao produtor, que responde com uma mensagem ao consumidor. As mensagens são definidas de uma forma que sejam compreensíveis tanto para o consumidor quanto para o produtor de serviços. Um produtor também pode ser um consumidor de um outro serviço (FERREIRA, 2015).

Figura 1 – Arquitetura básica SOA



Fonte: (FERREIRA, 2015)

### Web Services

A comunicação entre computadores em uma rede evoluiu bastante nos últimos anos. Desde o uso de soquetes de rede, formados basicamente pela combinação de um endereço IP (*Internet Protocol*) e um número de uma porta lógica, passando por arquiteturas como a CORBA (*Common Object Request Broker Architecture*), orientada a objetos e classes, até finalmente chegar em uma arquitetura como a SOA (*Service-oriented Architecture*),



baseada em serviços, as tecnologias sempre buscaram melhorias, facilitando a comunicação entre dispositivos. Como exemplo disso, as duas últimas arquiteturas citadas (CORBA e SOA) permitem a troca de informações entre diferentes plataformas, independentemente de seus sistemas operacionais e linguagens de programação, uma característica essencial para o desenvolvimento da IoT.

Nas décadas de 60 e 70, os *softwares* eram pensados como ferramentas de apoio específicas para determinadas rotinas, sendo assim, não havia uma unificação entre todo o sistema de uma ampla organização e todos esses pequenos sistemas executando tarefas isoladas, demandam elevados custos de manutenção (RIBEIRO; FRANCISCO, 2016).

Dentro dessa estrutura de TI (Tecnologia da Informação), cada sistema tinha um conjunto de dados e informações próprias, porém, devido ao fato de os processos de negócios envolverem diversas áreas dentro dessa organização, elas passaram a demandar serviços de mais de um sistema, o que tornou necessária a troca de informações entre eles (FUGITA; HIRAMA, 2012).

"Neste universo de sistemas distribuídos, o termo serviço pode ser conceituado como a 'execução de um trabalho ou realização de uma função de um prestador para um requisitante'. A comunicação entre estas duas partes (requisitante e servidor) se propaga superando obstáculos geográficos e heterogeneidades tecnológicas, trafegando por um meio de comunicação onipresente e disponível para todas as partes envolvidas: a Web. Seria, portanto, um *Web service*." (RIBEIRO; FRANCISCO, 2016).

Para isso, a comunicação com os *web services* foi padronizada, o que permitiu criar *web services* independentes da plataforma usada nos programas que solicitam um serviço. Esta padronização foi definida usando os protocolos HTTP (*Hypertext Transfer Protocol*) na transmissão dos dados (MAGRI, 2013).

### 2.2.1 Serviços web baseados em SOAP

O SOAP (*Simple Object Access Protocol*) é um protocolo para transferir dados através da internet. Ele se baseia em chamada de acesso remoto, e suas chamadas são representadas em XML (*Extensible Markup Language*). Esse protocolo usa um conceito de envelope que possui *header* e *body*. O *header* é um elemento opcional na mensagem, ele serve para especificar os requisitos a nível do aplicativo, o *body* é um elemento obrigatório que contém os dados que estão sendo trocados na mensagem SOAP. Ele deve ser posto, na estrutura do envelope, após o *header*, caso este exista (FERREIRA, 2015).

### 2.2.2 Serviços web baseados em REST

Representational State Transfer, ou REST, é um conjunto coordenado de restrições arquitetônicas que tenta minimizar latência e comunicação de rede, enquanto ao mesmo

tempo maximizar a independência e escalabilidade de implementações de componentes. Isso é conseguido colocando restrições na semântica do conector, onde outros estilos se concentraram na semântica de componentes. O REST permite o armazenamento em cache e reutilização de interações, a substituíbilidade dinâmica de componentes e o processamento de ações por intermediários, de forma a atender às necessidades de um sistema hipermídia distribuído na Internet (FIELDING; TAYLOR, 2002).

A arquitetura REST é uma implementação da SOA, tendo surgido como alternativa a esse estilo. Diferentemente da SOA tradicional, que usa o protocolo SOAP, se comunicando por mensagens em XML, o REST utiliza o protocolo HTTP para a comunicação com mensagens em JSON (*JavaScript Object Notation*), o que deixa as mensagens com menor volume de dados, portanto, mais leves. Sendo assim, mais adequadas para dispositivos com pouca memória e baixo poder de processamento, como sistemas embarcados (FIELDING, 2000).

JSON é um formato de texto para a serialização de dados estruturados, ou seja, transformar esses dados em um formato que possa ser armazenado. É derivado dos literais de objeto de JavaScript. O JSON pode representar quatro tipos primitivos (*strings*, *numbers*, *booleans* e *null*) e dois tipos estruturados (*objects* e *arrays*). Uma *string* é uma sequência de zero ou mais caracteres Unicode. Um objeto é uma coleção não ordenada de zero ou mais pares nome/valor, onde um nome é uma *string* e um valor é uma *string*, *number*, *boolean*, *null*, *object* ou *array*. Um *array* é uma sequência ordenada de zero ou mais valores. Os termos *object* e *array* vêm das convenções de JavaScript. Os objetivos de design do JSON eram para ele ser mínimo, portátil, textual e um subconjunto de JavaScript (CROCKFORD, 2006). JSON é em formato texto e completamente independente de linguagem, pois usa convenções que são familiares às linguagens C e familiares, incluindo C++, C#, Java, JavaScript, Perl, Python e muitas outras. Estas propriedades fazem com que JSON seja um formato fácil para troca de dados<sup>2</sup>.

REST é um estilo de arquitetura baseada em serviços que usa o HTTP para fazer chamadas entre máquinas. Não requer muitos recursos, pois trabalha com os padrões já definidos pelo HTTP. O protocolo de comunicação é chamado de mensagem REST, ela é dividida em Verbo, URI (*Uniform Resource Identifier*), HTTP *version*, *request header* e *request body*. O verbo é um dos métodos HTTP, como o GET, PUT, POST, DELETE, OPTIONS, entre outros. Cada verbo é exemplificado na Tabela 1 e eles servem como uma identificação da função da mensagem, encontrando-se logo no início. O substantivo, que é a URI, representa o recurso no qual a operação vai ser realizada. O HTTP *version* indica o tipo de HTTP da mensagem (FERREIRA, 2015).

<sup>2</sup> JSON. <<http://json.org/json-pt.html>>.

Tabela 1 – Métodos do REST

Método	Descrição
GET	Solicita uma representação específica de um recurso.
PUT	Cria ou atualiza um recurso com a representação fornecida.
DELETE	Elimina o recurso especificado.
POST	Submete dados a serem processados pelo recurso identificado.
HEAD	Semelhante ao GET, mas só recupera o <i>header</i> e não o <i>body</i> .
OPTIONS	Retorna os métodos suportados pelo recurso identificado.

Fonte – (FERREIRA, 2015)

Devido ao seu desempenho e suas características citadas, diversas empresas vêm adotando o protocolo REST. As plataformas de Amazon, Yahoo e Google, entre outras, oferecem soluções baseadas em REST (chamadas RESTful), pelas vantagens de escalabilidade e menor peso, além da tendência estar apontando para um menor uso de dados e protocolos, para dispositivos mais básicos, simplificando o estabelecimento da comunicação (MENDES, 2014).

## 2.3 Trabalhos Relacionados

Utilizando conceitos parciais da técnica de revisão sistemática da literatura, a partir de bases como IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos) Explore e ACM Digital Library (*Association for Computing Machinery*), em conjunto com o portal Google Acadêmico, utilizando palavras chaves como "SOA", "*Web Services*", "*Embedded Systems*", "*Mobile*" e "REST", pôde-se chegar a um conjunto de artigos que abordam aplicações da tecnologia escolhida, nas mais diversas plataformas de hardware.

### 2.3.1 Estudo Comparativo de Tecnologias

Estudo comparativo das tecnologias em plataformas embarcadas

1. Schall, Aiello e Dustdar (2006)<sup>3</sup> propõem o uso de *web services* em dispositivos embarcados para resolver problemas de interoperabilidade em sistemas distribuídos mobile. Discutem-se vários *toolkits* (conjunto de ferramentas) disponíveis para dispositivos embarcados e investiga a performance de *web services* em *smartphones*. A meta é guiar o projeto de aplicações *mobile* baseadas em *web services*, e oferecer estimativas de performance que podem ser esperadas. Foi provida uma visão geral de diferentes *toolkits* de *web services* disponíveis para a plataforma Symbian. O estudo compara a performance de *web services*

<sup>3</sup> Obs.: Os artigos estão ordenados por ano, do mais antigo ao mais recente. A numeração serve para referenciá-los na Tabela 2.

baseados em Java/J2ME (*Micro Edition*), com *toolkits* disponíveis na linguagem de programação C++. Discutiui-se um *framework* para obter medidas de performance online como, por exemplo, o tempo de processamento da mensagem. Acredita-se que *web services* em dispositivos embarcados têm um papel importante no acesso ou fornecimento de conteúdo multimídia em dispositivos móveis. Os resultados mostram tempo de requerimento menor usando gSOAP/C++ (*toolkit* C/C++ para *web services* SOAP/XML).

2. O artigo de Tergujeff et al. (2007) nos mostra o antigo ambiente móvel onde há alguns anos a arquitetura vem sendo aplicada, principalmente na integração de diversos ambientes para facilidade de troca de dados e aprimoramento do cotidiano, na era dos telefones com *firmwares* proprietários de cada marca, a predominância de linguagem de programação era o JAVA J2ME, devido ao suporte comum entre esses sistemas a aplicações nesta linguagem, existem bibliotecas que tratam da comunicação dos *web services* com esses antigos dispositivos como a kSOAP, essas tecnologias foram indicadas na criação de uma programa que integrasse editores textuais de *desktops* com celulares para uma maior praticidade de usuários de escritório, o ambiente antigo era desafiador, devido a necessidade de um endereço de IP e outras informações onde os dispositivos se mostravam ineficientes, necessitando ainda um aperfeiçoamento das técnicas.

3. Spiess et al. (2009) propõem uma arquitetura para uma integração eficiente de Internet da Coisas em serviços empresariais. A arquitetura foi implementada, baseada em padrões *open WS-\** (como *WS-Addressing* e *WS-Security*), em Java EE 5, na plataforma NetWeaver, usando EJB 3.0 (*Enterprise JavaBeans*) e JPA (*Java Persistence API*). Utiliza arquitetura de integração orientada a serviços SOCRADES (*Service-Oriented Cross-layer infrastructure for Distributed smart Embedded devices*). Espera-se que dispositivos heterogêneos sejam conectados a essa arquitetura, como dispositivos industriais, domésticos, sistemas de TI, celulares, PDAs (*Personal Digital Assistant*), máquinas de produção, robôs, sistemas de automação de prédios, carros, sensores e atuadores, leitores RFID, leitores de código de barras, ou medidores de energia. Os autores afirmam que a maioria das funcionalidades especificadas já foram implementadas e testadas. No entanto, os autores planejam conduzir testes de performance e escalabilidade, com foco especial em performance enquanto roda DPWS (*Devices Profile for Web Services*) em dispositivos *resource-constrained*. Trabalhos futuros irão incluir uma extensão da arquitetura para suportar dispositivos adicionais usando *plug-ins* OPC-UA (*Open Platform Communications Unified Architecture*) e REST. Também se planejam testes de validação com dispositivos do mundo real no domínio de automação e energia.

4. Shelby (2010) dá uma visão geral sobre arquitetura web, seu núcleo em REST, e o andamento atual de *web services*. Apresenta atividades para se alcançar *web services* eficientes em dispositivos embarcados, e introduz a tecnologia de comunicação CoRE (*Constrained RESTful Environment*), que está sendo desenvolvida utilizando o núcleo de

funcionalidades REST e pretende padronizar um protocolo para *web services* embarcados. De acordo com o autor, aplicando-se os fundamentos de REST no domínio de *web services*, é possível alcançar interfaces eficientes de *web services* entre dispositivos embarcados, mas para isso é necessária uma nova abordagem em relação a formatos usados em protocolos de transferência. Avanços recentes na encodificação de XML, por meio da tecnologia EXI (*Efficient XML Interchange*), têm mostrado resultados promissores.

5. Guinard et al. (2011) descrevem a arquitetura WoT (*Web of Things*), práticas baseadas nos princípios REST, e discute vários protótipos que usam esses princípios. Ele defende que devido à simplicidade da arquitetura REST e à grande variedade de bibliotecas e clientes HTTP, a arquitetura REST é uma das plataformas de integração mais leves. Para comunicação *machine-to-machine*, JSON tem se mostrado uma alternativa mais leve em comparação a XML, portanto, acredita que é mais adequado para dispositivos com capacidades limitadas. Por isso, esse conjunto de tecnologias deve ser utilizado como padrão em interações entre *Smart Things*.

6. Eliasson et al. (2013) propõem implantar IoT em parafusos de ancoragem em minas subterrâneas, permitindo o monitoramento em tempo real do status de sensores acoplados aos parafusos, detectando anomalias nas rochas e prevendo possíveis desabamentos e abalos sísmicos, aumentando, assim, a segurança dos trabalhadores. Na arquitetura sugerida, a comunicação é feita usando 6LoWPAN (*IPv6 over Low power Wireless Personal Area Networks*) e CoAP, a integração é baseada em SOA, o servidor pode se comunicar com os dispositivos por texto, binário, XML ou EXI. Os seguintes componentes da arquitetura foram implementados com sucesso: sistema de medidas, integração dos parafusos, plataforma de sensores sem fio e a arquitetura SOA. Apesar disso, segundo o artigo, pesquisas ainda precisam ser feitas, mas as conquistas alcançadas mostram que a proposta é factível e realista.

7. Zanella et al. (2014) mostram uma pesquisa sobre as tecnologias de base, protocolos, e arquitetura para uma IoT urbana. Apresenta e discute soluções técnicas e práticas usadas no projeto *Padova Smart City*, uma prova de conceito de IoT urbana na cidade de Pádua, na Itália. Para maior integração entre uma diversidade de IoT *nodes*, sugere tecnologias tanto para *unconstrained* quanto para *constrained*. Para *Unconstrained IoT nodes*, recomenda o uso de HTML/XML (*HyperText Markup Language*) para mensagem, HTTP/TCP para protocolo de comunicação, e IPv4/IPv6 (*Internet Protocol version 4/Internet Protocol version 6*) como tecnologia de rede. Para *Constrained IoT nodes*, utiliza mensagem EXI, comunicação CoAP/UDP (*Constrained Application Protocol*)/(User Datagram Protocol), e rede IPv6/6LoWPAN.

8. Catarinucci et al. (2015) propõem uma arquitetura IoT com o objetivo de monitorar e rastrear pacientes, funcionários e aparelhos médicos em hospitais e institutos de enfermagem. O sistema dispõe de tecnologias diferentes, porém, complementares, como

RFID (*Radio-Frequency Identification*) e *smartphones*, interoperando entre si através do protocolo CoAP/IPv6 por meio de uma rede sem fio de baixo consumo de energia 6LoWPAN/REST. Os resultados atingidos demonstram que o sistema proposto é apropriado para os objetivos.

9. Para [Júnior e Soares \(2015\)](#), a aplicação da arquitetura SOA se mostrou ideal para os objetivos de criação de um robô que tivesse todos os seus componentes acessíveis separadamente, ela traz a vantagem de fraco acoplamento para os componentes, podendo esses serem fácil programados com a linguagem JAVA, devido a uma biblioteca já existente, Myrobotlab, que trabalha conceitos de robótica com SOA, através de uma interface gráfica intuitiva.

10. [Ferreira \(2015\)](#) demonstra que com o aperfeiçoamento dos *softwares* móveis, temos hoje uma maior unificação dos *softwares* usados, as que tiveram maior sucesso no mercado são a Apple IOS e o Google Android, porém o desafio continua o mesmo, a busca pela comunicação unificada entre as plataformas, os mesmos conceitos de *web services* são aplicados, porém visando uma melhor performance, surge uma simplificação da arquitetura SOA, o REST que comprovadamente mostra uma mais rápida resposta de solicitação, por utilizar uma semântica de comunicação mais simples e objetiva, e como a área móvel embora muito avançada hoje em dia, continua limitada, soluções mais rápidas são ideais.

11. [Nazim, Shah e Abbasi \(2016\)](#) descrevem vários modelos de WoT e sugerem tecnologias de comunicação ideais para cada caso. Para a integração de dispositivos embarcados, foi sugerida a arquitetura SOA, por ser amigável para integração de aplicações, uma arquitetura de *web services* baseada em HTTP para trocar mensagens XML, usando SOAP. A pesquisa explora diferentes *web services* e arquiteturas para uma WoT embarcada. O artigo afirma também que uma rede integrada de dispositivos oferece a vantagem de adaptar-se facilmente a dispositivos novos, reduzir custos e integrar verticalmente organizações.

### Estudo comparativo das tecnologias em servidores

REST apresenta recursos como a captação primária de informação e funcionalidade. Recursos em repouso são abordados através URLs usando o HTTP e seus métodos para acessá-los. A ideia básica por trás do conceito do REST é que o estado de um recurso, por exemplo, um dispositivo, pode ser obtido ou definido pelos métodos HTTP GET ou POST, respectivamente.

[Chang, Mohd-Yasin e Mustapha \(2009\)](#) [1] <sup>4</sup> implementaram um protótipo de *Web Server RESTful* embarcado na placa FPGA (*Field Programmable Gate Array*) e testado em ambiente de 100Mbps LAN (*Local Area Network*), destacando nos resultados o quanto

<sup>4</sup> Esta numeração serve para referenciar os artigos na [Tabela 3](#)



o estilo REST para os *web services* simplifica as mensagens, tornando-as mais curtas e, por consequência, exigindo menos recursos do servidor.

O protótipo de Luckenbach et al. (2005) [2] foi desenvolvido com uma placa de sensores MTS310 e o microcontrolador dedicado à comunicação MICAz. Trifa et al. (2009) [4] implementaram tanto o servidor quanto o cliente focados em testes de desempenho. Outro caso focado na análise de desempenho desse tipo de aplicação foi desenvolvido por Nunes et al. (2014) [5] que elaboraram cliente e servidor em dispositivos Raspberry Pi, além de fazerem uma parte dos testes utilizando o ambiente de emulação QEMU (*Quick Emulator*).

Abeele et al. (2013) [9] utilizaram CoAP como protocolo de comunicação para seus testes em cenários de automação residencial. CoAP <sup>5</sup> é um protocolo que foi oficialmente documentado em 2014 e é baseado em REST. Ele é especializado em transferência web para ser utilizado em nós limitados e redes restritas na Internet das Coisas.

Machado et al. (2006) [3] que implementaram *web services* SOAP/WSDL (*Web Services Description Language*) em uma placa SHIP baseada em ARM usando o gSOAP toolkit. Leu, Chen e Hsu (2014) [6] utilizam um extenso conjunto de ferramentas para criar e testar uma rede de dispositivos que se comunicam via *web services*. Os autores testaram o desempenho do sistema tanto em relação ao envio e recebimento das mensagens quanto ao trabalho com diferentes *browsers* e principalmente a sua performance e eficácia no ambiente de Internet das Coisas.

O protocolo SOAP é, normalmente, associado a aplicações mais robustas, como expõe Shelby (2010), porém Bucci et al. (2005) [7] mostraram que mesmo esse tipo de aplicação pode ser desenvolvida com custo (tanto financeiro quanto computacional) reduzido. Perumal et al. (2013) [8] propõem um sistema gerenciador para automação residencial baseado no protocolo de comunicação SOAP/XML. Essa proposta expõe a proximidade e facilidade de trabalhar Internet das Coisas com a arquitetura orientada a serviços.

### 2.3.2 Tabelas comparativas

A Tabela 2 é composta pelos pontos principais de cada artigo selecionado na revisão. É separada nas colunas Plataforma, onde são mostrados os ambientes em que as arquiteturas estão sendo desenvolvidas, Arquitetura de Serviços, onde está qual a arquitetura presente naquele ambiente e, por fim, Padrão de Mensagem, onde é mostrado o tipo de mensagem trocada no projeto.

---

<sup>5</sup> COAP. <<http://coap.technology/>>

Tabela 2 – Resumo da análise de tecnologias para plataformas embarcadas e móveis

Nº	Plataforma	Arq. de serviços	Padrão de Mensagem
1	Mobile	SOA	XML
2	Mobile	SOA	SOAP
3	Várias/indefinidas plataformas	SOA	Java EE 5
4	Sistemas Embarcados	REST	XML
5	Várias/indefinidas plataformas	REST	JSON
6	Rock bolts	SOA	XML
7	IoT nodes (RFtags), Mobile devices	REST	XML
8	RFID tag, Smart Mobile	REST	-
9	Desktop e Arduino	SOA	SOAP
10	Mobile Android	REST	JSON
11	Sistemas Embarcados	SOA	XML

Fonte – Os autores, 2017

A [Tabela 3](#) é composta pelas principais tecnologias citadas nos artigos sobre servidores, a tabela está dividida em colunas que mostram as arquiteturas adotadas no servidor, o protocolo de comunicação e, por fim, o servidor usado no desenvolvimento de cada projeto.

Tabela 3 – Resumo da análise de tecnologias para servidor

Artigo	Plataforma Arquitetura de Serviços	Arq. de serviços Protocolo de Comunicação	Padrão de Mensagem Servidor
1	REST	RESTful	Spartan-3E Starter Board (embarcado)
2	REST	TinyRest	Laptops
3	SOA	SOAP	SHIP Board (Embarcado)
4	REST	Restlet	Laptops
5	REST	RESTFul	Raspberry Pi (Embarcado)
6	SOA	SOAP	Placa baseada em ARM11 (Embarcado)
7	SOA	SOAP	PIC18F452 (Embarcado)
8	SOA	SOAP	SHIP Board (Embarcado)
9	REST	CoaP	Alix System Board (Embarcado)

Fonte – Os autores, 2017



### 2.3.3 Análise do Estudo Comparativo

De acordo com a discussão de cada artigo, os pontos apontados indicam que a tecnologia REST é a mais adequada para integração de todos os dispositivos, pois tem-se nas tabelas 2 e 3 vários dispositivos em que essa tecnologia é empregada. Por usar os métodos do HTTP com uma mensagem mais enxuta, ela é mais leve e mais fácil de manipular nos dispositivos pretendidos, pois os protocolos HTTP já são facilmente encontrados por padrão em diversos meios. Para [Ferreira \(2015\)](#),

"Podemos verificar alguns fatos. As mensagens SOAP têm um maior volume de dados (as mensagens estão representadas em XML), em contraste as mensagens REST são o oposto, tornando-as mais adequadas para dispositivos que requeiram menor memória, como o caso dos *smartphones*. O REST identifica os recursos apenas com uma URL, utiliza qualquer um dos verbos HTTP (GET, POST, etc.) para realizar operações, ao contrário do SOAP que apenas usa o HTTP POST. Com o REST podemos armazenar a informação em cache com o HTTP GET, com o SOAP tal não é possível."([FERREIRA, 2015](#))

Portanto, a tecnologia de serviço que melhor se adequa às necessidades do projeto é o REST com troca de mensagem em JSON. Pois o trabalho envolve dispositivos com capacidade de processamento e conexão limitadas, além de que uma solução de conexão mais completa se torna dispensável para um projeto de pequena aplicação.

### 3 Metodologia

A execução deste projeto requer uma diversidade de áreas que interagem e englobam conhecimentos sobre tecnologias que atendem os requisitos para a definição de uma arquitetura para Internet das Coisas. Portanto, a primeira fase deste projeto se concentrou em realizar uma revisão da literatura. Esta, juntamente com uma pesquisa sobre ferramentas e soluções tecnológicas disponíveis dentro do universo da proposta, pautou a escolha das tecnologias que serão utilizadas. O desenvolvimento de sistemas embarcados que necessitem interagir entre si possibilitará a realização de uma prova de conceitos para experimentar a arquitetura proposta.

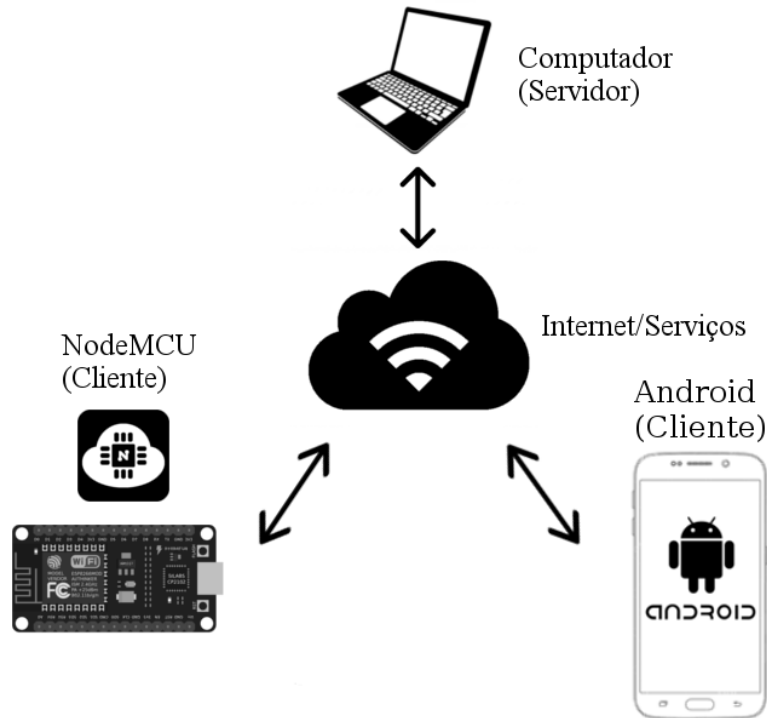
Neste âmbito, este projeto, como um todo, requer expertises em diferentes áreas técnicas como por exemplo eletrônica, algoritmos e métodos formais da engenharia de software para construção do sistema de informação aqui proposto. A metodologia adotada contempla os seguintes passos:

1. Revisão da literatura;
2. Seleção das técnicas e linguagens que implementam sistemas baseados em serviços;
3. Definição da Arquitetura Orientada a Serviços de baixo custo para Internet das Coisas;
4. Projeto e desenvolvimento de uma prova de conceito com sistemas embarcados integrados via arquitetura proposta;
  - 4.1. Definição e escolha da prova de conceito;
  - 4.2. Definição do hardware a ser adotado nas diversas camadas;
  - 4.3. Configuração e testes do servidor de serviços;
  - 4.4. Testes de comunicação de serviços REST com Bancos de Dados;
  - 4.5. Implementação da prova de conceito;
  - 4.6. Testes e coleta de dados da prova de conceito.
5. Análise dos resultados da prova de conceito;
6. Escrita do documento final.

Conforme o objetivo e a fundamentação teórica, a arquitetura descrita para a montagem é melhor apresentada na [Figura 2](#), onde o servidor estará executando em uma máquina que tenha suporte a esta tecnologia, podendo esta máquina ser de um computador

peçoal até um Raspberry Pi, onde ela irá disponibilizar serviços que serão consumidos pelo Android e pela placa NodeMCU.

Figura 2 – Visão geral da arquitetura



Fonte: Os autores, 2017

### 3.1 Servidor

GlassFish <sup>1</sup> é um servidor de aplicações baseado na Plataforma Java e tecnologia *Enterprise Edition* (Java EE) para o desenvolvimento e execução de aplicações e serviços web, oferecendo desempenho, confiabilidade, produtividade e facilidade. Atualmente ele se encontra na versão 4.1.1, suas características fortes para montar um *web service* estão em alta disponibilidade, escalabilidade e tolerância a falhas, que são fundamentais para um servidor de produção que hospede aplicações de médio e grande porte (LUVIZON, 2012).

Um sistema de banco de dados é basicamente um sistema computacional de armazenamento e manutenção de registros, em outras palavras, é um sistema que armazena informações e permite que os usuários as busquem e modifiquem. Estes sistemas estão disponíveis em máquinas que variam de *smartphones* até mainframes ou *clusters* de computadores de grande porte (DATE, 2004).

<sup>1</sup> Glassfish - Pesquisas de compiladores. <<http://pesquompile.wikidot.com/glassfish>>.

## 3.2 Cliente Embarcado

O ESP8266 é um chip de baixo custo da fabricante chinesa Espressif Systems que possui o protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*) integrado. A primeira versão o chip (ESP-01) servia apenas como um módulo para conectar microcontroladores a uma rede *WiFi*. Hoje em dia, as novas versões, como a ESP-12, já possuem uma Unidade MicroControladora (MCU) incluída na placa ([HACKADAY, a](#)) ([HACKADAY, b](#)).

NodeMCU é uma plataforma IoT de código aberto. Ela inclui o *firmware* que roda no ESP8266 e um *hardware* baseado no módulo ESP-12. O *firmware* usa a linguagem de programação Lua. A plataforma NodeMCU foi escolhida pelo seu baixo custo e por possuir os recursos suficientes para diversas aplicações que envolvam a Internet das Coisas. Este dispositivo é o mais adequado para o presente projeto pois o sistema proposto deve possuir um baixo custo e realizar uma comunicação entre sensores e um servidor, e a plataforma NodeMCU desempenha esta função. Uma outra característica importante para a escolha do NodeMCU foi a capacidade de programação utilizando o IDE (*Integrated Development Environment*) do Arduino, o qual permite usar uma variação da linguagem C (linguagem Wiring) ([SILVEIRA; LEITE, 2016](#)).

## 3.3 Cliente Móvel

O desenvolvimento de um cliente de testes será com uma aplicação móvel desenvolvida para o sistema operacional Android, que é um sistema operacional móvel que foi desenvolvido como uma solução completa para celulares, possuindo já pacotes com programas para celular, aplicativos, interface do usuário e controle de hardware e software, sendo hoje a plataforma mais utilizada em smartphones, para ([PEREIRA; SILVA, 2009](#)),

"O Android foi construído com a intenção de permitir aos desenvolvedores criar aplicações móveis que possam tirar total proveito do que um aparelho portátil possa oferecer. Foi construído para ser verdadeiramente aberto. Por exemplo, uma aplicação pode apelar a qualquer uma das funcionalidades de núcleo do telefone, tais como efetuar chamadas, enviar mensagens de texto ou utilizar a câmera, que permite aos desenvolvedores adaptarem e evoluírem cada vez mais essas funcionalidades"

O seu ponto forte está na capacidade de fazer com que todos os dispositivos que o possui, seguir padrões de conexões, processamento e programação, facilitando o desenvolvimento de novos aplicativos que aprimoram o seu uso.

## 4 Prova de Conceito

O sistema no qual é aplicada a proposta de IoT utilizando serviços é uma estrutura de monitoramento de ambiente por sensores de presença e temperatura. Em intervalos definidos, o sensor de temperatura é ativado, no projeto foi usado um intervalo de cinco minutos, e cada vez que o sensor de presença é ativado, ou seja, cada vez que ele detecta um movimento, são enviadas as informações para o servidor através de um método HTTP POST, essas informações contêm o número do sensor que está sendo ativado e, no caso do sensor de temperatura, a temperatura em graus Celsius. A mensagem é passada no formato JSON e deve seguir os padrões determinados pelo servidor.

Ao receber a mensagem pelo POST, o servidor se encarrega de verificar se é um JSON válido, caso não seja, ele retorna uma mensagem de erro, caso for válido, ele faz o tratamento do JSON e armazena seus dados em um banco de dados. Os dados armazenados são a data e a hora do recebimento, o número do sensor e, se for o caso, a temperatura. Cada operação é feita com um serviço diferente, há um serviço POST para o sensor de temperatura e um para o de presença.

O aplicativo para Android faz uma solicitação GET no servidor, que responde de acordo com a informação solicitada, assim como para o POST, o servidor também possui dois serviços GET, um para temperatura e outro para presença. Ao fazer essa solicitação, é retornado um JSON com todos o histórico de eventos armazenados do sensor ou temperatura, então o Android faz o tratamento desse dado e mostra na tela o último evento armazenado para temperatura ou presença.

Nas próximas subseções (Servidor, Cliente Embarcado, e Cliente Móvel), são mostradas instruções de como reproduzir a prova de conceito produzida para este trabalho. Instruções mais detalhadas podem ser encontradas nos Apêndices deste documento. O [Apêndice A](#) descreve a criação de servidor com banco de dados, o [Apêndice B](#) descreve os procedimentos para conectar os sensores à placa NodeMCU, e o conjunto ao servidor. O [Apêndice C](#) descreve como criar um aplicativo para plataformas móveis que receba os dados do sistema proposto. Os resultados da prova de conceito são mostrados no fim deste capítulo, na seção Resultados.

### 4.1 Servidor

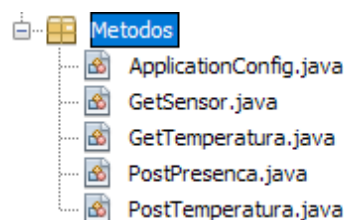
Para o gerenciamento da estrutura, o servidor REST foi desenvolvido com o programa GlassFish, junto ao NetBeans.

Os serviços disponibilizados pelo servidor são:

- Armazenar dados do sensor de presença: um método POST que recebe um JSON com o número do sensor, seguido o exemplo `{"sensornumero":"1"}`. O método armazena este número do sensor, junto com a hora e data do servidor, quando ele recebeu essa solicitação;
- Armazenar dados do sensor de temperatura: um método POST que recebe um JSON com o número do sensor e a temperatura, seguido o exemplo `{"temperatura":"38.94","sensornumero":"1"}`. O método armazena o valor da temperatura e o número do sensor de temperatura, junto com a hora e data do servidor, quando ele recebeu essa solicitação;
- Mostrar histórico de dados do sensor de presença: um método GET que, ao ser solicitado, retorna todos os eventos em que o sensor de presença foi ativado, a mensagem é um JSON que segue o padrão exemplo `{"data":"2016-12-16","horas":"01:39:31","sensor numero":"1"}`;
- Mostrar histórico de dados do sensor de temperatura: um método GET que, ao ser solicitado, retorna todos os eventos em que o sensor de temperatura foi ativado, a mensagem é um JSON que segue o padrão exemplo `{"temperatura":"30.97","data":"2016-12-26","horas":"21:49:05","sensornumero":"1"}`.

Cada serviço desenvolve seus métodos em uma classe diferente no NetBeans, como mostrado na [Figura 3](#).

Figura 3 – Classes dos serviços

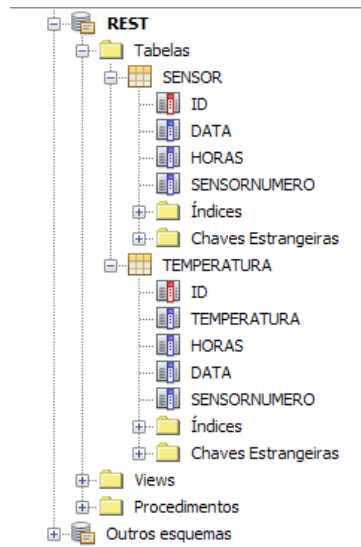


Fonte: Os autores, 2017

#### 4.1.1 Banco de dados

O servidor trabalha com um banco de dados SQL chamado Derby, para os serviços, foram criadas duas tabelas que são mostradas na [Figura 4](#), uma para cada sensor, que armazenam essas informações de forma estruturada.

Figura 4 – Tabelas do banco de dados



Fonte: Os autores, 2017

O bando de dados Derby pode ser utilizado na forma de servidor ou embarcado, na forma de servidor, o BD (Banco de Dados) deve estar rodando em algum lugar, geralmente na máquina, e o acesso é feito pela URL dele: `localhost:"porta"/"nome do banco de dados"`, já na forma embarcada, o acesso é feito pelo caminho da pasta onde o BD está armazenado. No servidor, o banco de dados foi usado na forma embarcada, então foi transferida a pasta de dados para a mesma do projeto do *web service*.

#### 4.1.2 Trabalhando e prevenindo erros com JSON

Para o gerenciamento de mensagem JSON no servidor, foi usada a biblioteca do Google chamada Gson <sup>1</sup>, e foi utilizada no projeto a versão 2.3.1, ela é adicionada como um arquivo .jar.

Dentre as funções da biblioteca, a utilizada na aplicação foi a conversão de JSON em objetos Java, e vice-versa, como mostra a Figura 5.

Figura 5 – Convertendo dados para JSON

```
@Path("getTemperatura")
public class GetTemperatura {

    @GET
    @Produces("application/json")
    public String getGreeting() {
        GerenciarDAO gd = new GerenciarDAO();
        Gson g = new Gson();
        return g.toJson(gd.getTemperatura());
    }
}
```

Fonte: Os autores, 2017

<sup>1</sup> GitHub. Gson User Guide. <<https://github.com/google/gson/blob/master/UserGuide.md>>.

As mensagens JSON no servidor devem seguir um padrão para que a conversão seja possível, portanto, são utilizadas várias camadas de verificação para mensagens recebidas, como mostra a [Figura 6](#).

Figura 6 – Tratamento de erros com dados recebidos

```
@POST
@Consumes("text/json")
public String setName(String content) {
    try {
        Gson g = new Gson();
        Sensor s = g.fromJson(content, Sensor.class);
        if (s.getSensornumero() == null) {
            return "JSON inválido | Padrão exemplo: {\\"sensornumero\\":\\"1\\"}";
        } else {
            GerenciarDAO gd = new GerenciarDAO();
            gd.updatePresenca(s.getSensornumero());
            return "É um JSON válido, movimento registrado para o sensor: " + s.getSensornumero();
        }
    } catch (JsonSyntaxException e) {
        return "Não é um JSON | Padrão exemplo: {\\"sensornumero\\":\\"1\\"}";
    }
}
```

Fonte: Os autores, 2017

As duas verificações que o servidor faz, são: verificar se o dado recebido não é nulo, ou seja, possui um valor, ou se o JSON recebido está no padrão do servidor. Caso ele não obedeça uma dessas regras, é retornada uma mensagem de erro junto com explicações do padrão que deve ser adotado.

### 4.1.3 Funcionamento do acesso

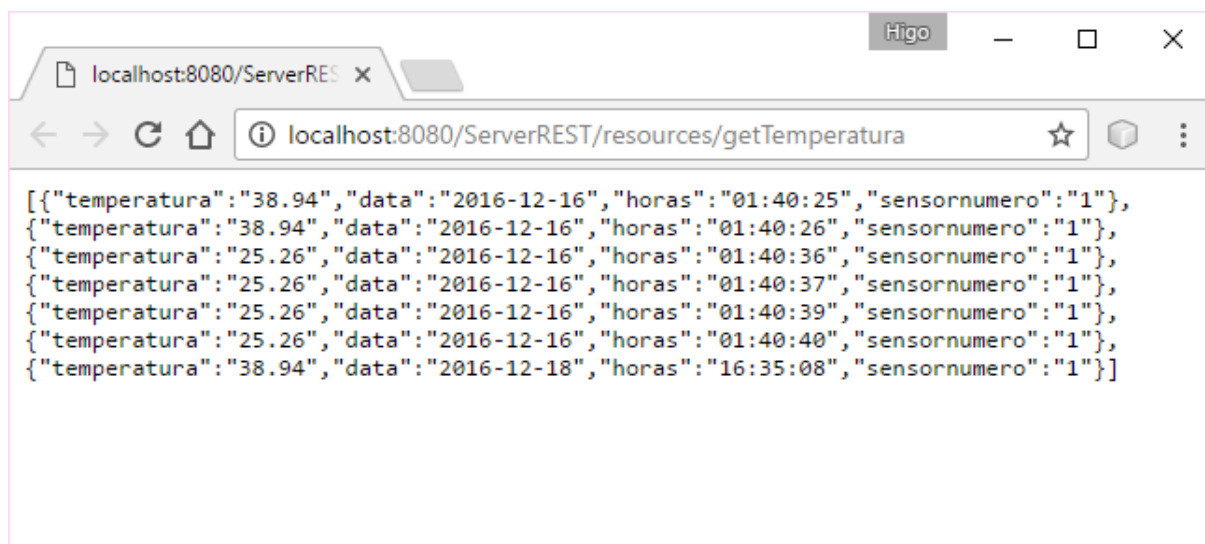
O acesso é feito através de uma URL, cada serviço tem sua própria, a estrutura da URL é definida da seguinte forma:

- localhost ou IP da máquina local: é o endereço de IP local da rede que a máquina rodando o *server* obtém;
- 8080: porta que é feita a troca de informação;
- Nome do servidor: no caso do que foi criado, tem o nome de ServerREST, é definido com o nome criado no início do projeto;
- Application Path: esse caminho é definido e pode ser alterado na classe Application-Config.java, é um identificador de servidor;
- Path: é o caminho definido em cada classe.



A [Figura 7](#) exemplifica o acesso do método GET, que pode ser feito através de um navegador.

Figura 7 – Requisição GET



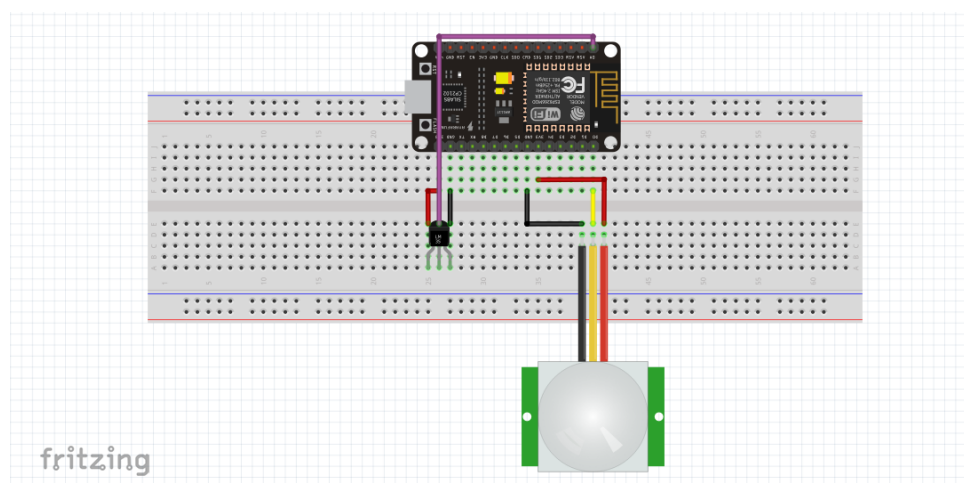
Fonte: Os autores, 2017

O tutorial detalhado de criação do servidor é encontrado no [Apêndice A](#).

## 4.2 Cliente Embarcado

Na aplicação embarcada, foram utilizados uma placa NodeMCU, um sensor de temperatura e um sensor de movimento. A conexão dos sensores na placa foi feita em uma *Protoboard*. O esquema elétrico da conexão é mostrado na [Figura 8](#).

Figura 8 – Esquema elétrico



Fonte: Os autores, 2017

Para o código que roda no NodeMCU, foram utilizados alguns comandos presentes do exemplo de código "ESP8266HTTPClient: BasicHttpClient", disponível no IDE do Arduino, no menu Arquivo -> Exemplos -> ESP8266HTTPClient -> BasicHttpClient. Antes da função "*setup*", são incluídas as bibliotecas necessárias, declaradas as variáveis auxiliares e são nomeados os pinos do NodeMCU onde os sensores serão conectados, como mostrado na [Figura 9](#).

Figura 9 – Comandos prévios

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266HTTPClient.h>
#define USE_SERIAL Serial

ESP8266WiFiMulti WiFiMulti;

const int SensorTempPin = A0;
const int SensorMovPin = 16;
int count = 0;
int estado = 0;
String ip = "192.168.0.2";
//String ip = "jhs.hopto.org";|
```

Fonte: Os autores, 2017

Na função "*setup*"([Figura 10](#)), no comando:

```
WiFiMulti.addAP("SSID", "PASSWORD");
```

Define-se o nome e a senha da rede *WiFi* que o módulo deverá se conectar, trocando a palavra "SSID" (*Service Set Identifier*) pelo nome da rede, e "PASSWORD" pela senha, mantendo as aspas em ambos os casos. Exemplo:

```
WiFiMulti.addAP("JH", "12345678");
```

Caso o servidor que será acessado não tenha um domínio público *online*, é necessário conectar na mesma rede *WiFi* que o computador com o servidor está conectado.

Figura 10 – Função "setup"

```
void setup() {  
  USE_SERIAL.begin(115200);  
  USE_SERIAL.println();  
  USE_SERIAL.println();  
  USE_SERIAL.println();  
  
  for (uint8_t t = 4; t > 0; t--) {  
    USE_SERIAL.printf("[SETUP] WAIT %d...\n", t);  
    USE_SERIAL.flush();  
    delay(1000);  
  }  
  
  WiFiMulti.addAP("JH", "12345678");  
  
  pinMode(SensorTempPin, INPUT);  
  pinMode(SensorMovPin, INPUT);  
  Serial.begin(115200);  
}
```

Fonte: Os autores, 2017

Na função "loop" é colocado o algoritmo que será executado. No presente experimento, a seguinte lógica foi empregada: A medição da temperatura é feita a cada intervalo de tempo predefinido, e a cada medição, a mensagem é enviada para o servidor pelo método POST. O sensor de movimento, ao ser ativado, envia instantaneamente uma mensagem para o servidor pelo mesmo método. O código completo é encontrado no [Apêndice B](#), no fim deste documento.

Substitui-se o endereço no comando "http.begin" pelo endereço para o qual o programa enviará mensagens pelo método POST. A URL é encontrada manualmente no NetBeans. Exemplos:

```
http.begin("http://" + ip + ":8080/ServerREST/resources/postPresenca");
```

```
http.begin("http://" + ip + ":8080/ServerREST/resources/postTemperatura");
```

Observa-se que os serviços responsáveis por cada medição (do sensor de temperatura e do de movimento) são independentes.

Feitas todas estas modificações, o programa está pronto. Ao ser enviado para a placa, o processo de conexão do módulo é iniciado, os dados dos sensores são lidos e enviados para o servidor indicado por meio do método POST.

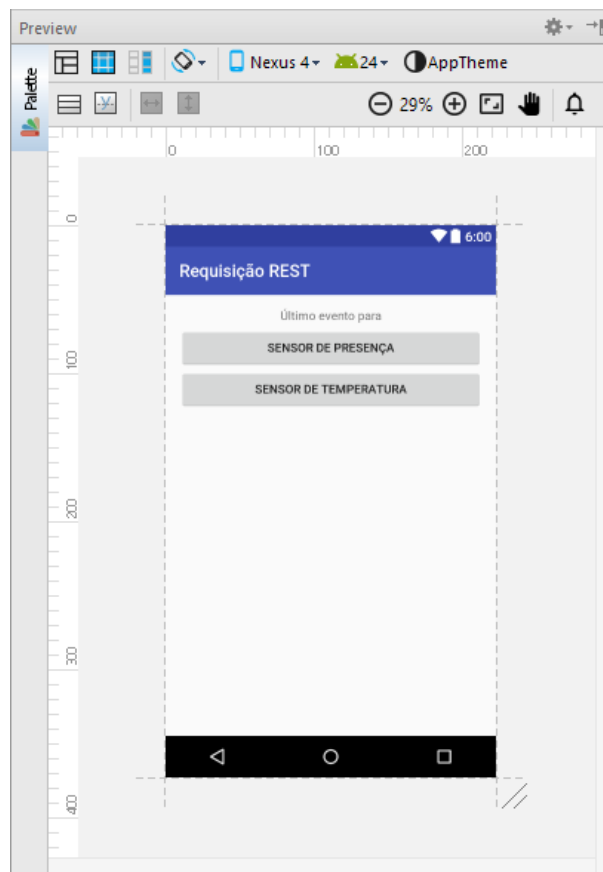
O tutorial de montagem e codificação completa do NodeMCU é encontrado no [Apêndice B](#).

### 4.3 Cliente Móvel

Visando a parte que utiliza os dados disponíveis no servidor, um breve exemplo foi desenvolvido. Os serviços, para serem consumidos, necessitam de uma plataforma, no caso, é utilizada a plataforma Android, a proposta central é o desenvolvimento de um aplicativo consumidor desses serviços, que trabalhe na mesma arquitetura que o servidor, portanto, um aplicativo que faça uma requisição GET no servidor, pegue a resposta em JSON, manipule os dados e exiba a informação na tela do dispositivo.

Para isso, o aplicativo foi desenvolvido no IDE do Android Studio, criado pela Google para criação de aplicativos utilizando as linguagens Java para programação e XML para design. O design e funcionalidade do aplicativo foram pensados apenas como forma de exemplificação, não demonstrando todos os recursos possíveis de serem desenvolvidos com os dados obtidos. O aplicativo mostra, ao ser solicitado, o último evento armazenado no servidor de cada sensor. São mostrados na tela o número do sensor que foi feito o registro, a data e hora do evento e, caso o sensor solicitado foi o de temperatura, o valor dela também é exibido, em graus Celsius. A interface do aplicativo é mostrado na [Figura 11](#). O tutorial de criação mais detalhado da aplicação Android é encontrado no [Apêndice C](#).

Figura 11 – Layout XML do aplicativo

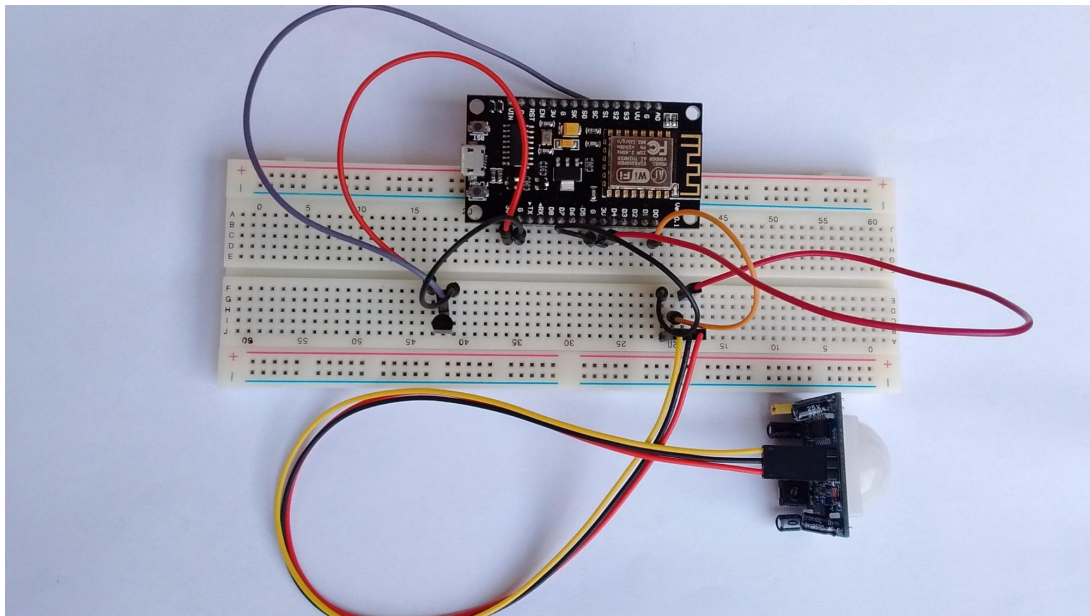


Fonte: Os autores, 2017

## 4.4 Resultados

A placa NodeMCU foi montada em uma *Protoboard* (Figura 12), conectando-se os sensores em suas determinadas portas de entrada, como planejado no esquema elétrico (Figura 8). A placa é alimentada com a tensão de 5V provenientes da entrada USB (*Universal Serial Bus*) de um computador, e alimenta os sensores com a tensão de 3,3V. A alimentação também poderia ser feita utilizando-se um carregador de celular, que transforma a tensão da tomada de 110/220V para 5V na saída micro-USB. Os preços dos componentes utilizados, e o total gasto para a construção desta prova de conceito são apresentados na Tabela 4 <sup>2</sup>.

Figura 12 – Montagem na Protoboard



Fonte: Os autores, 2017

Tabela 4 – Preços dos componentes

Componente	Preço
Módulo WiFi ESP8266 NodeMcu ESP-12E	R\$ 59,90
Sensor de Movimento Presença PIR DYP-ME003	R\$ 12,90
Sensor de Temperatura LM35	R\$ 8,90
Total	R\$ 81,70

Fonte – Os autores, 2017

<sup>2</sup> Valores retirados do site FilipeFlop.

<<http://www.filipeflop.com/pd-2c140d-modulo-wifi-esp8266-nodemcu-esp-12e.html>>;

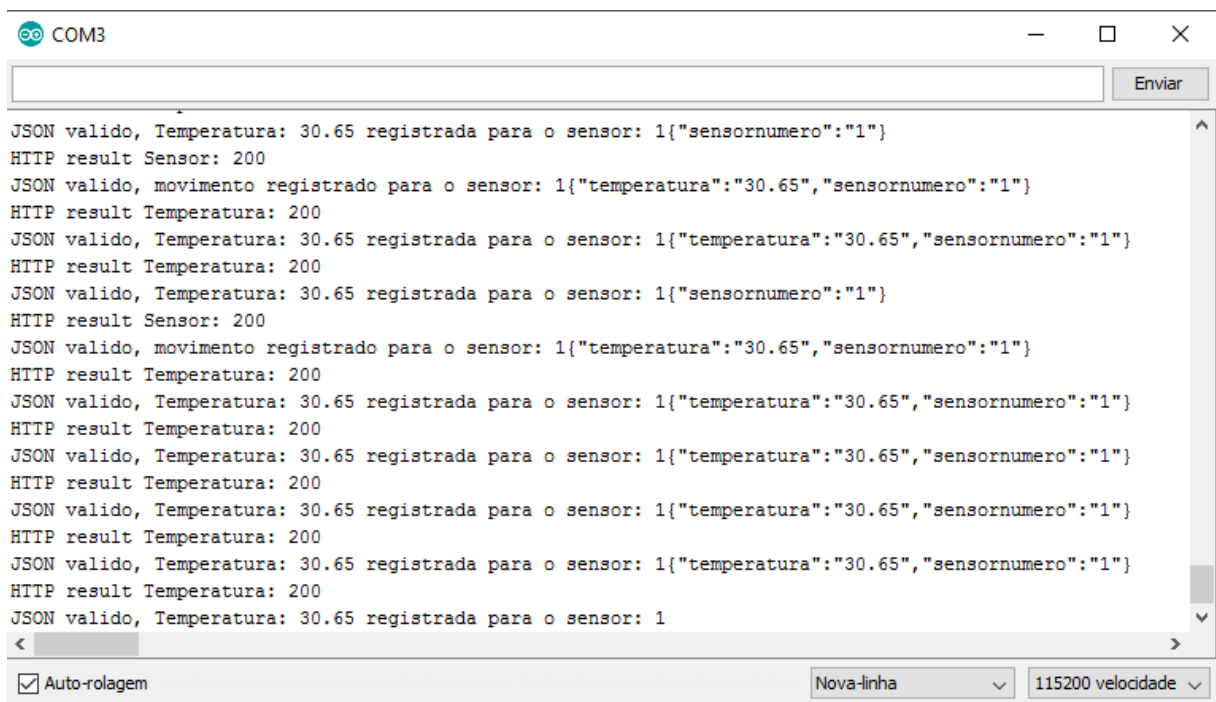
<<http://www.filipeflop.com/pd-6b901-sensor-de-movimento-presenca-pir.html>>;

<<http://www.filipeflop.com/pd-22565c-sensor-de-temperatura-lm35dz.html>>.

O teste do sistema foi realizado em uma sala climatizada, portanto, era possível alterar a temperatura do ambiente para verificar a eficiência da medição realizada pelo sensor de temperatura, e se os dados provenientes dos sensores eram enviados corretamente. O sensor de presença tem sua ativação de forma binária, ou seja, está ativado ou desativado, ele contém dois potenciômetros para ajustar a distância de medição e o tempo de aviso. Os ajustes foram feitos de forma satisfatória para detectar os movimentos de uma pessoa em uma sala. O conjunto de sensores ficaram em uma posição superior ao nível médio da altura da sala, para evitar a obstrução de detecção por objetos e o sensor de temperatura foi ajustado para mandar uma medição de temperatura a cada cinco minutos.

O NodeMCU foi configurado para se conectar a uma rede *WiFi* na qual o servidor também está conectado. Então, foi ajustado o envio das informações dos sensores para o IP local da máquina do servidor, de acordo com a URL de cada serviço. O resultado de cada medição e o resultado do envio dos dados para o servidor pode ser visualizado na [Figura 13](#).

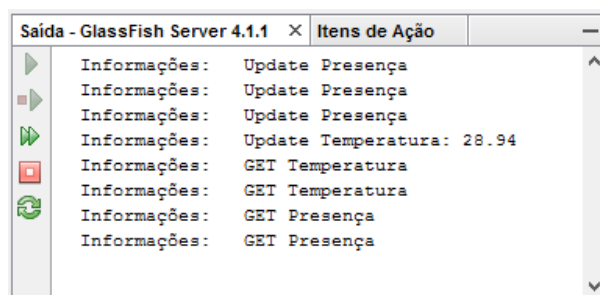
Figura 13 – Serial



Fonte: Os autores, 2017

No servidor, instalado em um computador com o sistema operacional Windows, sendo executado no programa NetBeans, foi verificado o recebimento e registro dos dados pela saída do programa, onde foi configurado para mostrar cada vez que um serviço fosse executado, conforme a [Figura 14](#).

Figura 14 – Recebimento dos dados pelo servidor



Fonte: Os autores, 2017

A mensagem é exibida quando todos os recursos foram executados com sucesso, ou seja, apenas se a mensagem for uma mensagem JSON válida. O acesso ao banco de dados foi concedido e feito o armazenamento da informação. A visualização dos registros no banco de dados é mostrada na [Figura 15](#).

Figura 15 – Informações armazenadas do banco de dados

ID	TEMPERATURA	HORAS	DATA	SENSORNUMERO
610	28.71	21:42:12	2016-12-26	1
611	29.03	21:42:15	2016-12-26	1
612	28.71	21:42:17	2016-12-26	1
613	28.71	21:42:19	2016-12-26	1
614	28.71	21:42:21	2016-12-26	1

Fonte: Os autores, 2017

O aplicativo foi testado em três *smartphones* Android com configurações diferentes, tendo resultados satisfatórios em todos, pois foi executado sem erros. O interface final do aplicativo de teste é apresentado na [Figura 16](#).

Figura 16 – Interface do aplicativo



Fonte: Os autores, 2017



## 5 Considerações finais

A prova de conceito, como um todo, funcionou de forma coerente com o que foi proposto, conseguiu-se aplicar uma arquitetura orientada a serviços em dispositivos embarcados, fazendo a integração deles. Os dados do monitoramento são disponibilizados através de serviços HTTP, com mensagem em JSON, seguindo os padrões de um servidor REST, podendo ser acessado por qualquer dispositivo que tenha conhecimento da URL e trabalhe com os padrões de comunicação, sendo possível tal dispositivo fazer a manipulação dos dados da forma que ele achar necessário.

Há uma série de vantagens visíveis ao se aplicar serviços na troca de dados entre sistemas embarcados. Em comparação com o exemplo feito no projeto, sem serviços, a disponibilização dos dados da placa NodeMCU teria que ser pensada de forma independente para cada plataforma. Para o Android, por exemplo, o tipo de comunicação teria que ser feita via Bluetooth <sup>1</sup>, e tais dados seriam visíveis apenas por um dispositivo conectado por vez, não sendo possível a criação do banco de dados em outras plataformas. Portanto, percebe-se a universalidade dos dados independentemente das plataformas, como pretendido.

Certas funcionalidades que seriam de interesse para a expansão do projeto podem ser implementadas em trabalhos futuros. Funcionalidades voltadas à facilidade de uso, por exemplo, uma interface de seleção de redes *WiFi* poderia ser feita para conectar o NodeMCU às redes visíveis, sem a necessidade de configurar a conexão diretamente no código-fonte; o servidor poderia ser registrado com um IP público para a fixação da URL de cada serviço; Uma camada de segurança que encripte os dados poderia ser criada para proteger dados e serviços que são transmitidos, pois não são públicos; Um melhoramento na estratégia de manutenção também poderia ser criado, para que haja a facilidade de adicionar novos serviços futuramente; o sensor de movimento utilizado poderia ser um sensor baseado na tecnologia sonar, em vez de um baseado em infravermelho, o que permitiria a geração de dados analógicos em vez de digitais, possibilitando a identificação da quantidade de movimento lido, tornando viável a distinção do número de pessoas em um ambiente.

A aplicação móvel poderia ser aprimorada com mais funcionalidades que demonstrem a grande variedade de possibilidades de utilização dos dados recebidos. Poderia ser criado um algoritmo que calcule estatísticas relacionadas à frequência de ativação do sensor de presença, assim como consultas por períodos de tempo, ou seja, em vez de exibir somente o último dado recebido, exiba uma média dos dados recebidos em um determinado

---

<sup>1</sup> Arduino e Cia. <<http://www.arduinoecia.com.br/2014/01/comunicacao-arduino-android-com-bluetooth.html>>.

período de tempo. Tais estatísticas advindas dos dados podem ser utilizadas para traçar um perfil do usuário, caracterizando-se como processamento de dados, princípios das propostas IoT e *Big Data*.

As tecnologias adotadas foram o suficiente para o funcionamento do sistema desenvolvido, futuras aplicações podem mostrar limitações da arquitetura que não foram percebidas, por isso, se faz necessário o constante aperfeiçoamento dos processos e testes com serviços mais complexos.

# Referências

- ABEELE, F. Van den et al. Building embedded applications via rest services for the internet of things. In: ACM. *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*. [S.l.], 2013. p. 82.
- 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.
- BUCCI, G. et al. A low cost embedded web service for measurements on power system. In: IEEE. *IEEE Symposium on Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2005*. [S.l.], 2005. p. 6–pp.
- CATARINUCCI, L. et al. An iot-aware architecture for smart healthcare systems. *IEEE Internet of Things Journal*, IEEE, v. 2, n. 6, p. 515–526, 2015.
- CHANG, C.; MOHD-YASIN, F.; MUSTAPHA, A. An implementation of embedded restful web services. In: IEEE. *Innovative Technologies in Intelligent Systems and Industrial Applications, 2009. CITISIA 2009*. [S.l.], 2009. p. 45–50.
- CROCKFORD, D. The application/json media type for javascript object notation (json). 2006.
- DATE, C. J. *Introdução a sistemas de bancos de dados*. [S.l.]: Elsevier Brasil, 2004.
- ELIASSON, J. et al. A soa-based framework for integration of intelligent rock bolts with internet of things. In: IEEE. *Industrial Technology (ICIT), 2013 IEEE International Conference on*. [S.l.], 2013. p. 1962–1967.
- EVANS, D. The internet of things. *How the Next Evolution of the Internet is Changing Everything, Whitepaper, Cisco Internet Business Solutions Group (IBSG)*, v. 1, p. 1–12, 2011.
- FERREIRA, P. B. V. *Arquitetura REST em smartphones Android*. Dissertação (Mestrado), 2015.
- FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, Irvine, 2000.
- FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, ACM, v. 2, n. 2, p. 115–150, 2002.
- FILIPEFLOP.COM. *Como programar o NodeMCU com IDE Arduino | Blog FILIPEFLOP*. 2017. Disponível em: <<http://blog.filipeflop.com/wireless/programar-nodemcu-com-ide-arduino.html>>. Acesso em: 11 Jan. 2017.
- FILIPEFLOP.COM. *Módulo WiFi ESP8266 NodeMcu ESP-12 - FILIPEFLOP Componentes Eletrônicos*. 2017. Disponível em: <<http://www.filipeflop.com/pd-2c140d-modulo-wifi-esp8266-nodemcu-esp-12e.html?ct=&p=1&s=1>>. Acesso em: 11 Jan. 2017.

- FILIPEFLOP.COM. *Sensor de Movimento Presença PIR - FILIPEFLOP Componentes Eletrônicos*. 2017. Disponível em: <<http://www.filipeflop.com/pd-6b901-sensor-de-movimento-presenca-pir.html?ct=&p=1&s=1>>. Acesso em: 11 Jan. 2017.
- FILIPEFLOP.COM. *Sensor de Temperatura LM35DZ - FILIPEFLOP Componentes Eletrônicos*. 2017. Disponível em: <<http://www.filipeflop.com/pd-22565c-sensor-de-temperatura-lm35dz.html?ct=&p=1&s=1>>. Acesso em: 11 Jan. 2017.
- FRIESS, P. *Internet of things: converging technologies for smart environments and integrated ecosystems*. [S.l.]: River Publishers, 2013.
- FUGITA, H. S.; HIRAMA, K. *SOA: Modelagem, análise e design*. [S.l.: s.n.], 2012.
- GUINARD, D. et al. From the internet of things to the web of things: Resource-oriented architecture and best practices. In: *Architecting the Internet of Things*. [S.l.]: Springer, 2011. p. 97–129.
- HACKADAY. *New Chip Alert: The ESP8266 WiFi Module (It's \$5) | Hackaday*. Disponível em: <<https://hackaday.com/2014/08/26/new-chip-alert-the-esp8266-wifi-module-its-5/>>. Acesso em: 23 Jan. 2017.
- HACKADAY. *An SDK for the ESP8266 WiFi Chip | Hackaday*. Disponível em: <<https://hackaday.com/2014/10/25/an-sdk-for-the-esp8266-wifi-chip/>>. Acesso em: 23 Jan. 2017.
- HE, H. *What Is Service-Oriented Architecture*. 2003. Disponível em: <<http://www.xml.com/pub/a/ws/2003/09/30/soa.html>>. Acesso em: 11 Jan. 2017.
- JÚNIOR, G.; SOARES, C. S. Proposta de um framework baseado em arquitetura orientada a serviços para a robótica. 2015.
- LEU, J.-S.; CHEN, C.-F.; HSU, K.-C. Improving heterogeneous soa-based iot message stability by shortest processing time scheduling. *IEEE Transactions on Services Computing*, IEEE, v. 7, n. 4, p. 575–585, 2014.
- LUCKENBACH, T. et al. Tinyrest-a protocol for integrating sensor networks into the internet. In: *Proc. of REALWSN*. [S.l.: s.n.], 2005. p. 101–105.
- LUVIZON, J. G. Segurança e desempenho em aplicações web utilizando jaas, glassfish e postgresql. Medianeira, 2012.
- MACHADO, G. B. et al. Integration of embedded devices through web services: Requirements, challenges and early results. In: IEEE. *11th IEEE Symposium on Computers and Communications (ISCC'06)*. [S.l.], 2006. p. 353–358.
- MAGRI, J. A. Criando e usando web service. *Augusto Guzzo Revista Acadêmica*, v. 1, n. 11, p. 166–183, 2013.
- MATTERN, F.; FLOERKEMEIER, C. From the internet of computers to the internet of things. In: *From active data management to event-based systems and more*. [S.l.]: Springer, 2010. p. 242–259.

- MENDES, M. A. d. S. Estilos arquiteturais web baseados em padrões abertos w3c. *Governança da internet e a atuação brasileira*, p. 85, 2014.
- NAZIM, M.; SHAH, M. A.; ABBASI, M. K. Analysis of embedded web resources in web of things. *Proceedings Appeared on IOARP Digital Library*, 2016.
- NUNES, L. H. et al. Análise de desempenho em dispositivos limitados e emulados estudo de caso: Raspberry pi e web services restful. 2014.
- PEREIRA, L.; SILVA, M. D. *Android para Desenvolvedores*. BRASPORT, 2009. ISBN 9788574524993. Disponível em: <[https://books.google.com.br/books?id=Nk\\\_4LQEACAAJ](https://books.google.com.br/books?id=Nk\_4LQEACAAJ)>.
- PERUMAL, T. et al. Development of an embedded smart home management scheme. *International Journal of Smart Home*, Science & Engineering Research Support Society, v. 7, n. 2, p. 15–26, 2013.
- RECOMMENDATION, Y. 2060. *Overview of Internet of Things*. ITU-T, Geneva, 2012.
- RIBEIRO, M.; FRANCISCO, R. Web services rest conceitos, análise e implementação. *Revista Educação, Tecnologia e Cultura-ETC*, v. 14, n. 14, 2016.
- ROBOTS, A. *General Helpful and Reference Info – AmazingRobots.net*. 2017. Disponível em: <<http://amazingrobots.net/resources/general-helpful-and-reference-info/>>. Acesso em: 11 Jan. 2017.
- SCHALL, D.; AIELLO, M.; DUSTDAR, S. Web services on embedded devices. *International Journal of Web Information Systems*, Emerald Group Publishing Limited, v. 2, n. 1, p. 45–50, 2006.
- SHELBY, Z. Embedded web services. *IEEE Wireless Communications*, IEEE, v. 17, n. 6, p. 52–57, 2010.
- SILVEIRA, R. M.; LEITE, S. de L. Sistema de controle de acesso baseado na plataforma nodemcu. 2016.
- SPIESS, P. et al. Soa-based integration of the internet of things in enterprise services. In: IEEE. *Web Services, 2009. ICWS 2009. IEEE International Conference on*. [S.l.], 2009. p. 968–975.
- SURYADEVARA, N. K.; MUKHOPADHYAY, S. C. Wireless sensor network based home monitoring system for wellness determination of elderly. *IEEE Sensors Journal*, IEEE, v. 12, n. 6, p. 1965–1972, 2012.
- TERGUJEFF, R. et al. Mobile soa: Service orientation on lightweight mobile devices. In: IEEE. *IEEE International Conference on Web Services (ICWS 2007)*. [S.l.], 2007. p. 1224–1225.
- TRIFA, V. et al. Design and implementation of a gateway for web-based interaction and management of embedded devices. *Submitted to DCOSS*, Citeseer, p. 1–14, 2009.
- ZANELLA, A. et al. Internet of things for smart cities. *IEEE Internet of Things Journal*, IEEE, v. 1, n. 1, p. 22–32, 2014.

# APÊNDICE A – Criação de Servidor REST com banco de dados

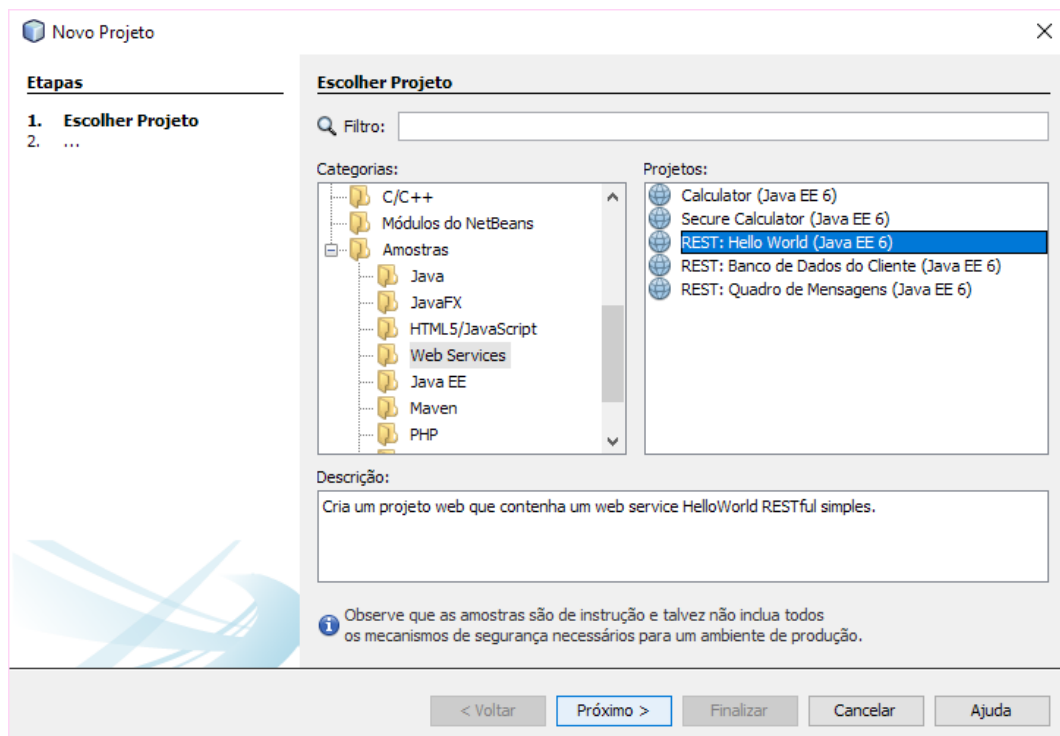
## A.1 Criando um Servidor

O servidor utilizado foi o GlassFish 4.1, devido a ele já vir como padrão com o IDE NetBeans 8.2 em seu pacote completo, ele foi escolhido para o desenvolvimento do servidor.

Para criar o novo projeto, devem-se seguir os seguintes passos com o NetBeans (Figura 17):

- No menu "arquivo", selecionar "novo projeto".
- Na tela de seleção, descer até a opção "amostras" e selecionar "Web Services".
- Na opção "projetos", selecionar "REST: Hello World".

Figura 17 – Tela de criação de projeto



Fonte: Os autores, 2017

A estrutura criada por padrão contém seis menus ([Figura 18](#)), que se diversificam entre códigos-fonte, páginas de web e configurações. A parte principal para a criação dos serviços consumidos está em pacotes de código-fonte, o projeto criado, como o título dele induz, cria por padrão um serviço que mostra um "Hello World" quando é dado o comando GET, esse serviço é desenvolvido na classe HelloWorldResource.java.

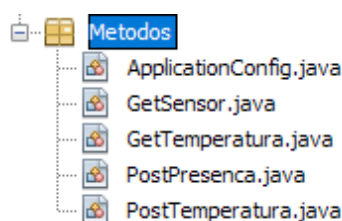
Figura 18 – Estrutura do projeto



Fonte: Os autores, 2017

Para o *server* que se pretende criar, são criadas quatro novas classes, nomeadas de `getTemperatura`, `getSensor`, `postPresenca` e `postTemperatura`, podendo-se apagar as outras classes que vieram como padrão, com a exceção da classe `ApplicationConfig.java` que, como o nome sugere, serve como uma classe "main" de configurações. Também é possível renomear o pacote a fim de uma melhor organização do projeto. As classes do projeto são exibidas na [Figura 19](#).

Figura 19 – Classes dos serviços



Fonte: Os autores, 2017

Apenas por questões de organização interna e melhor visualização, cada serviço foi posto em uma classe, também era possível criar uma classe GET e definir todos os serviços que usam esse método dentro dela, assim também como para o POST. Para configurar o caminho do serviço na URL de acesso é preciso definir na classe o nome do seu caminho, para isso usa-se `@Path("nome do serviço")` acima da declaração da classe, como mostra a [Figura 20](#).

Figura 20 – Caminho do serviço

```
*/  
  
@Path("getTemperatura")  
public class GetTemperatura {
```

Fonte: Os autores, 2017

Dentro da classe (Figura 21), é preciso definir o tipo de requisição que o servidor vai lidar, isso é feito com os chamados verbos do HTTP. No servidor são usados dois verbos, o GET e o POST, eles são declarados com @GET e @POST. Logo em seguida, é preciso definir um método para cada verbo, ou seja, indicar qual será o processamento das informações que cada um vai fazer. Também pode ser definido, antes do método, um tipo de dado que o servidor irá tratar, no caso deste servidor, ele trabalha apenas com dados de texto do tipo JSON, portanto, pode-se usar para o GET o @Produces("application/json") que especifica que ao ser solicitado, retorna um JSON. E para o POST, o @Consumes("text/json"), que define que a entrada é um JSON.

Figura 21 – Requisição

```
*/  
  
@Path("postPresenca")  
  
public class PostPresenca {  
  
    @POST  
    @Consumes("text/json")  
  
    public String setName(String content) {
```

Fonte: Os autores, 2017

## A.2 Criando banco de dados Derby

Também por já vir integrado por padrão na instalação do NetBeans completo e trabalhar de forma portátil, o banco de dados JavaDB (Derby) é indicado para aplicações que necessitem guardar dados de forma rápida e leve, como o caso do servidor deste trabalho.

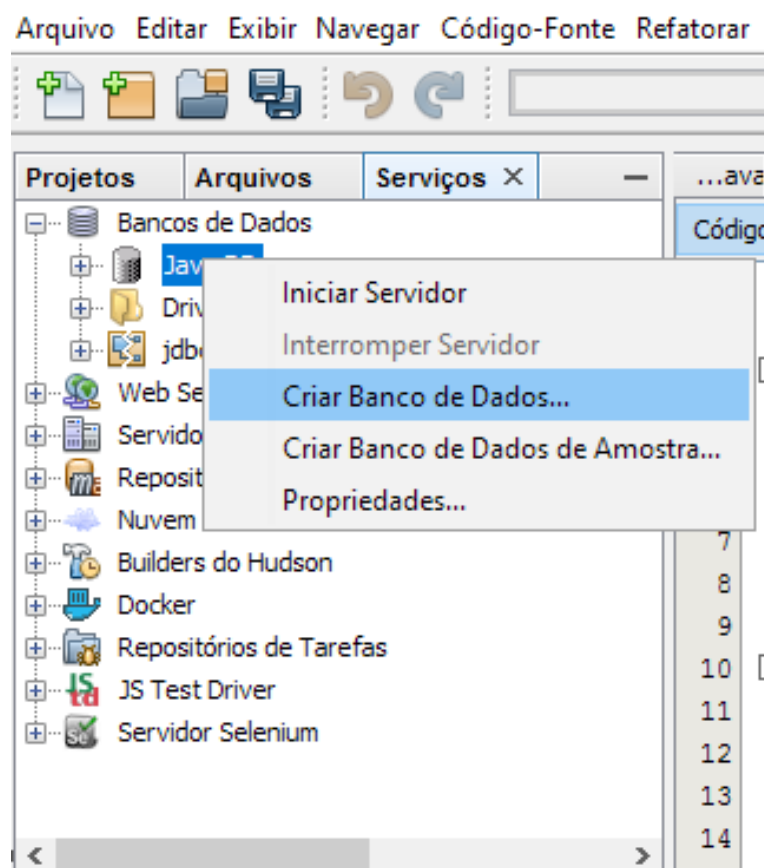
Para criar um banco de dados pelo NetBeans, deve-se seguir os seguintes passos (Figura 22):

- Ir na aba serviços.



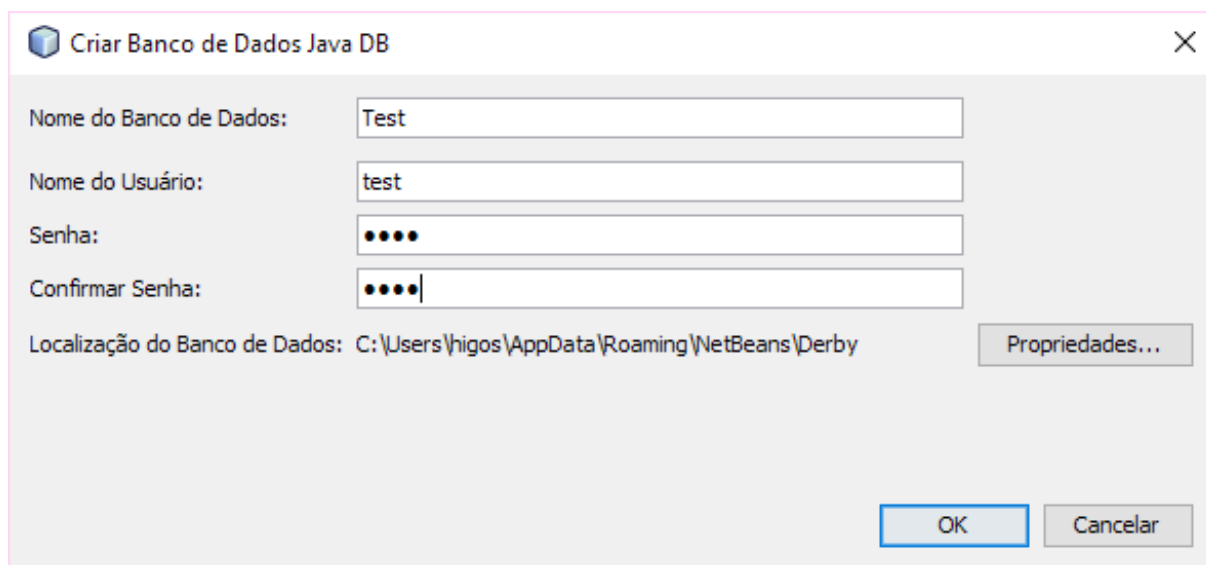
- Expandir a opção banco de dados.
- Clicar com o botão inverso ao *click* do mouse sobre "JavaDB".
- Selecionar a opção "Criar Banco de Dados..."

Figura 22 – Criando banco de dados



Fonte: Os autores, 2017

Na janela de criação é preciso definir o nome do banco de dados, um *login* e senha, e definir o local do disco onde será salvo esse banco de dados, como mostra a [Figura 23](#).

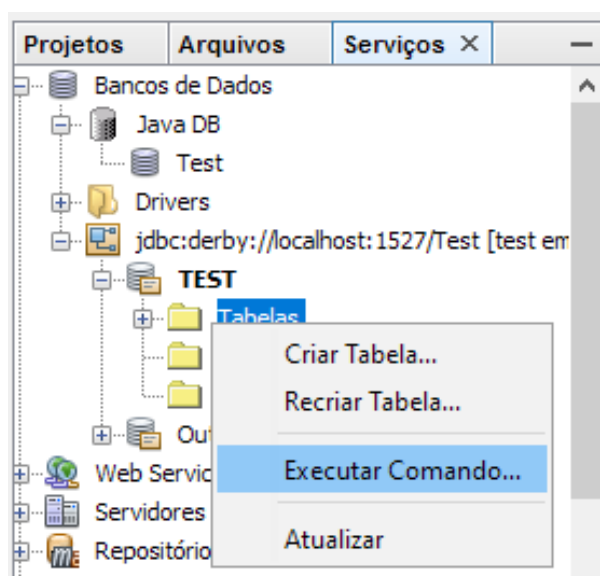
Figura 23 – Definindo *login* do banco de dados

Fonte: Os autores, 2017

Então o banco de dados criado estará disponível expandindo-se a opção "JavaDB" e clicando-se em "conectar".

Para gerenciar o banco de dados, há duas maneiras, por interface gráfica ou por comandos SQL (Figura 24), clicando-se com o botão inverso ao do *click* do mouse sobre as tabelas do BD, são apresentadas tais opções.

Figura 24 – Executar SQL

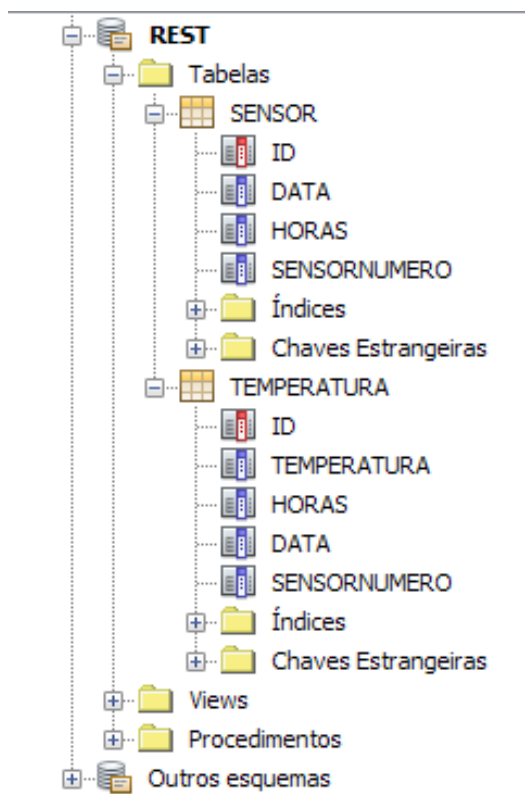


Fonte: Os autores, 2017

Para o servidor, foi criado um banco de dados com o nome REST e com duas tabelas e suas colunas, a tabela "sensor" com as colunas ID, Data, Horas e Sensornumero,

e a tabela "temperatura" com as colunas ID, Data, Horas, Temperatura e Sensornumero, conforme mostrado na [Figura 25](#).

Figura 25 – Tabelas do banco de dados



Fonte: Os autores, 2017

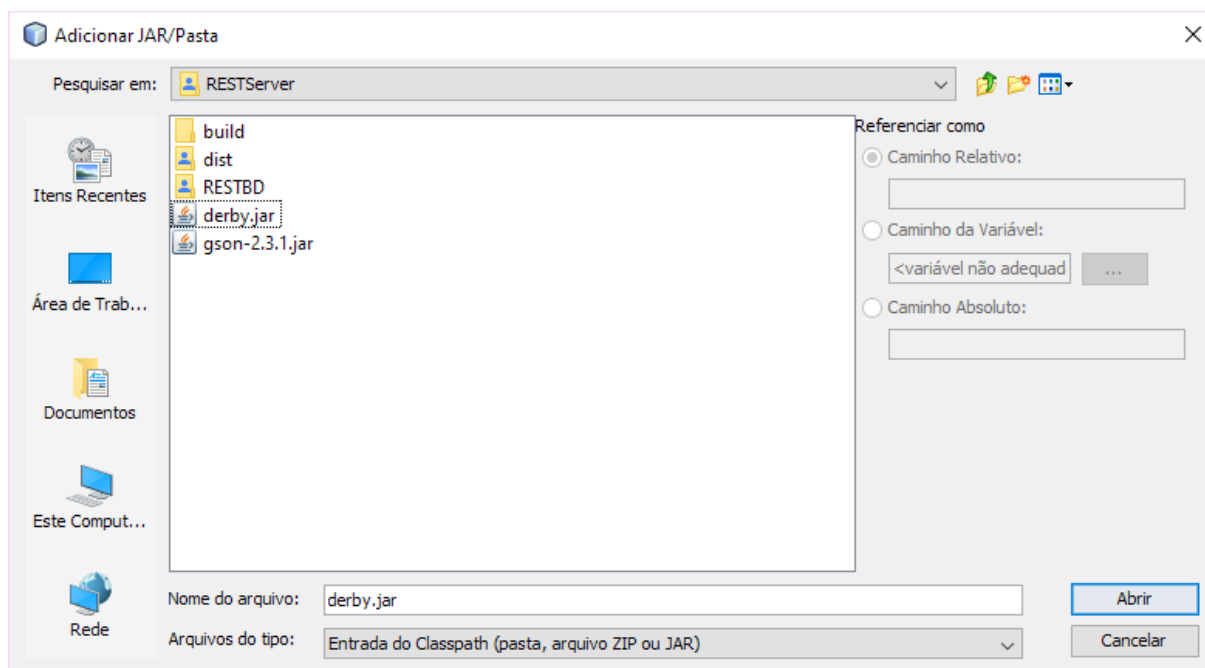
O bando de dados Derby pode ser utilizado na forma de servidor ou embarcado. Na forma de servidor, o BD deve estar rodando em algum lugar, geralmente na máquina, e o acesso é feito pela URL dele: localhost:"porta"/"nome do banco de dados", já na forma embarcada, o acesso é feito pelo caminho da pasta onde o BD está armazenado.

No servidor, o banco de dados foi usado na forma embarcada, então foi transferida a pasta de dados para a mesma do projeto do *web service*.

### A.3 Conectando o Servidor ao banco de dados

Para conectar o Servidor ao banco de dados, é preciso primeiro adicionar o drive desse banco ao projeto, isso é feito na opção biblioteca, apertando-se com o botão inverso ao do *click* e selecionando-se a opção Adicionar Jar/Pasta, onde irá aparecer um menu para selecionar o arquivo do *driver*, no caso, derby.jar ([Figura 26](#)).

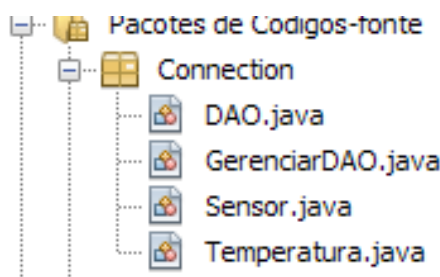
Figura 26 – Adicionando driver



Fonte: Os autores, 2017

Para essa conexão, foram criadas duas classes em um novo pacote, uma chamada "DAO" e outra chamada "GerenciarDAO", que são respectivamente uma classe para conexão e outra para gerenciamento de SQL. Também foram feitas duas classes para auxiliar na gerência dos objetos, a classe Temperatura e a classe Sensor, como mostra a [Figura 27](#).

Figura 27 – Classes de conexão



Fonte: Os autores, 2017

Na classe "DAO", estão armazenadas as informações do banco de dados, como: o caminho da pasta que ele se encontra, o *driver* utilizado, *login* e senha, e nela há métodos que abrem a conexão com o banco de dados ([Figura 28](#)).

Figura 28 – Classe DAO

```
public class DAO {  
    private static final String banco = "jdbc:derby:"  
        + "C:/Users/higos/Google Drive/UFMA/TCC/RESTServer/RESTBD";  
    private static final String driver = "org.apache.derby.jdbc.EmbeddedDriver";  
    private static final String usuario = "rest";  
    private static final String senha = "123";  
    private static Connection con = null;  
  
    public static Connection getConexao() {  
        if (con == null) {  
            try {  
                Class.forName(driver);  
                con = DriverManager.getConnection(banco, usuario, senha);  
            } catch (ClassNotFoundException ex) {  
                System.out.println("Não encontrou o driver");  
            } catch (SQLException ex) {  
                System.out.println("Erro ao conectar: "  
                    + ex.getMessage());  
            }  
        }  
        return con;  
    }  
  
    public static PreparedStatement getPreparedStatement(String sql) {  
        if (con == null) {  
            con = getConexao();  
        }  
        try {  
            return con.prepareStatement(sql);  
        } catch (SQLException e) {  
            System.out.println("Erro de sql: "  
                + e.getMessage());  
        }  
        return null;  
    }  
}
```

Fonte: Os autores, 2017

Já a classe "GerenciarDAO", cria métodos de inserção e obtenção de dados do banco de dados, nela que se criam os códigos SQL que farão as consultas de acordo com a finalidade, na [Figura 29](#) é mostrada uma consulta aos dados da tabela "sensor" do banco de dados.

Figura 29 – Classe GerenciarDAO

```
public class GerenciarDAO {  
  
    public ArrayList getSensor() {  
        ArrayList<Sensor> sensor = new ArrayList<Sensor>();  
        String sql = "SELECT * FROM sensor";  
        PreparedStatement pst = DAO.getPreparedStatement(sql);  
        ResultSet rs;  
        try {  
            rs = pst.executeQuery();  
  
            while (rs.next()) {  
                Sensor s = new Sensor();  
                s.setData(rs.getString("data"));  
                s.setHoras(rs.getString("horas"));  
                s.setSensornumero(rs.getString("sensornumero"));  
                sensor.add(s);  
            }  
            pst.close();  
        } catch (SQLException ex) {  
            Logger.getLogger(GerenciarDAO.class.getName()).log(Level.SEVERE, null, ex);  
        }  
        return sensor;  
    }  
}
```

Fonte: Os autores, 2017

O método `getSensor` na classe `GerenciarDAO`, pega todos os eventos guardados na tabela "sensor", ela guarda a SQL responsável por essa consulta na *String* de nome "sql", então chama o método `getPreparedStatement` da classe "DAO" e armazena seu retorno em uma variável "pst", o método chamado da classe DAO é responsável por verificar no método `getConexao` se ocorreu a conexão, então ele obtém a resposta para o SQL enviado e retorna esse dado para quem o chamou.

Os dados ficam armazenados em lista, então é percorrida essa lista retirando-se as informações de cada índice e adicionando-se a um objeto, sendo este, adicionado a uma *ArrayList* do mesmo tipo do objeto, todas as consultas de obtenção de dados seguem o mesmo princípio, como demonstra a [Figura 30](#).

Para a inserção de dados, o processo é o mesmo que para a conexão, porém, no SQL, são enviadas as informações para o banco de dados.

Figura 30 – Classe GerenciarDAO

```
public boolean updatePresenca(String sensor) {  
    String sql = "insert into sensor (sensornumero, data, horas)"  
        + " values (?,CURRENT_DATE,CURRENT_TIME)";  
    Boolean retorno = false;  
    System.out.print("Update Presença");  
    PreparedStatement pst = DAO.getPreparedStatement(sql);  
    try {  
        pst.setString(1, sensor);  
        if (pst.executeUpdate() > 0) {  
            retorno = true;  
        }  
    } catch (SQLException ex) {  
        Logger.getLogger(GerenciarDAO.class.getName()).log(Level.SEVERE, null, ex);  
        retorno = false;  
    }  
    return retorno;  
}
```

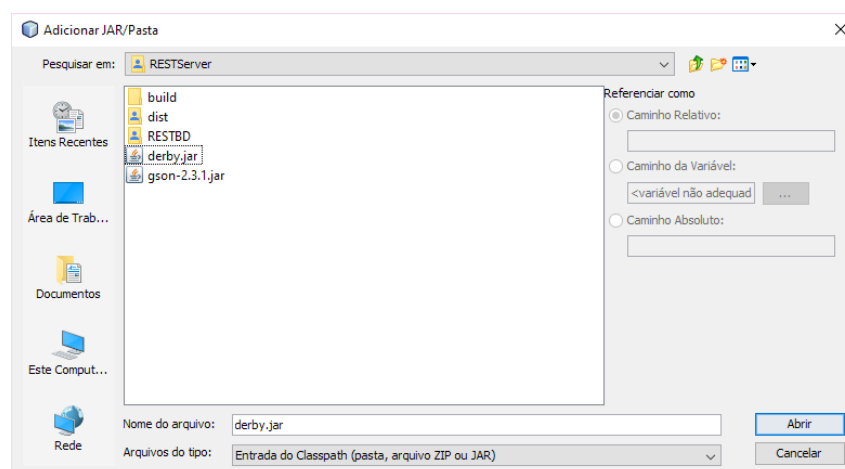
Fonte: Os autores, 2017

## A.4 Trabalhando com JSON

Para gerenciar dados em JSON, a Google disponibiliza uma biblioteca que se integra ao NetBeans, chamada Gson. Ela pode ser baixada gratuitamente e adicionada na opção biblioteca no projeto <sup>1</sup>.

Para adicionar a biblioteca Gson no servidor, é preciso primeiro adicionar o arquivo .jar no projeto, isso é feito na opção biblioteca, apertando-se com o botão inverso ao do *click* e selecionando-se a opção Adicionar Jar/Pasta, onde irá aparecer um menu para selecionar o arquivo do Gson, no caso, "gson-2.3.1.jar" (Figura 31).

Figura 31 – Adicionando biblioteca gson



Fonte: Os autores, 2017

<sup>1</sup> GitHub. Gson User Guide. <<https://github.com/google/gson/blob/master/UserGuide.md>>.

Para transformar dados em formato JSON, basta instanciar a classe Gson e então usar o método contido nela, chamado "toJson". Nos métodos GET, ela é responsável por transformar uma lista de dados advinda do banco de dados, pelos métodos da GerenciarDAO, em um JSON (Figura 32).

Nos métodos de POST ela é responsável por pegar um JSON e transformá-lo em uma classe com objetos dos dados desse JSON, usando o método "fromJson". O arquivo recebido deve estar no modelo da classe passada para que a transformação em objeto seja possível, qualquer informação fora do padrão, culminará em um erro.

Figura 32 – Convertendo JSON para objeto Java

```
public class PostTemperatura {  
  
    @POST  
    @Consumes("text/json")  
    public String setName(String content) {  
        try{  
            Gson g = new Gson();  
            Temperatura s = g.fromJson(content, Temperatura.class);  
        }  
    }  
}
```

Fonte: Os autores, 2017

## A.5 Funcionamento dos serviços

Cada classe e seu método são chamados por uma URL diferente, o método chamado na classe é aquele que está abaixo do @"Verbo"do HTTP, o GET ou POST.

Para as classes GET, sempre é necessário que o método tenha um "return", que vai ser a informação que ela vai responder ao ser chamada (Figura 33).

Figura 33 – Método GET

```
@Path("getSensor")  
public class GetSensor {  
  
    @GET  
    @Produces("application/json")  
    public String getGreeting() {  
        GerenciarDAO gd = new GerenciarDAO();  
        Gson g = new Gson();  
        return g.toJson(gd.getSensor());  
    }  
}
```

Fonte: Os autores, 2017

Para as classes POST, um método com um argumento de entrada é definido, nesse argumento será armazenada a informação de entrada enviada pelo POST, a classe não é



obrigada a ter uma informação de retorno, porém, há a possibilidade (Figura 34).

Figura 34 – Método GET

```
public class PostPresenca {  
  
    @POST  
    @Consumes("text/json")  
  
    public String setName(String content) {  
        try {  
            Gson g = new Gson();  
            Sensor s = g.fromJson(content, Sensor.class);  
            GerenciarDAO gd = new GerenciarDAO();  
            gd.updatePresenca(s.getSensornumero());  
            return "movimento registrado para o sensor: " + s.getSensornumero();  
        }  
    }  
}
```

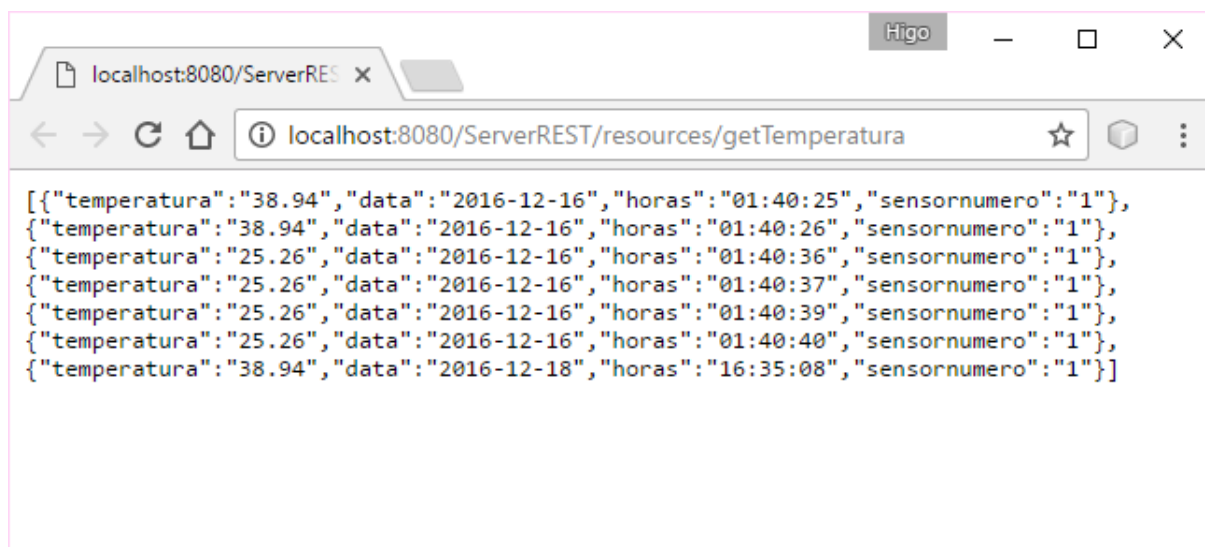
Fonte: Os autores, 2017

A URL de acesso para esse método é determinada da seguinte forma:

- localhost ou IP da máquina local: é o endereço de IP local da rede que a máquina rodando o *server* obtém.
- 8080: porta que é feita a troca de informação.
- Nome do servidor: no caso do servidor criando neste trabalho, tem o nome de "ServerREST", é definido com o nome criado no início do projeto.
- Application Path: esse caminho é definido e pode ser alterado na classe Application-Config.java, é um identificador de servidor.
- Path: é o caminho definido em cada classe.

O resultado do método GET é fácil ser visualizado, pois qualquer navegador web faz o método GET na URL, então, colocando-se a URL no navegador, são apresentadas as informações (Figura 35).

Figura 35 – Requisição GET



Fonte: Os autores, 2017

O método POST não pode ser acessado diretamente pela URL do navegador, pois ele precisa enviar um parâmetro para o servidor. Com programas que realizam esse método, é possível visualizar o funcionamento, como é mostrado na Figura 36.

Figura 36 – Requisição POST



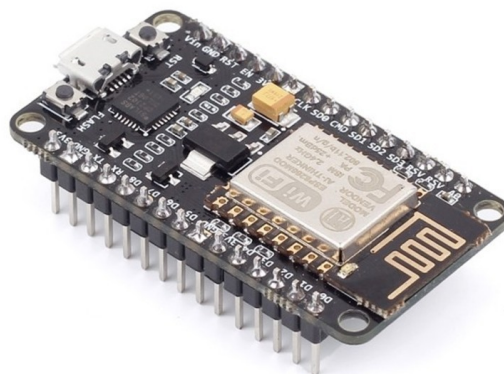
Fonte: Os autores, 2017

# APÊNDICE B – Procedimentos para conexão do módulo NodeMCU a um servidor REST

## Materiais Utilizados

- Módulo *WiFi* ESP8266 NodeMcu ESP-12E ([Figura 37](#))

Figura 37 – NodeMCU



Fonte: ([FILIPEFLOP.COM](#), 2017b)

### Especificações:

- Módulo NodeMcu Lua ESP-12E
- Wireless padrão 802.11 b/g/n
- Antena embutida
- Conector Micro-USB
- Modos de operação: STA/AP/STA+AP
- Suporta 5 conexões TCP/IP
- Portas GPIO: 11
- GPIO com funções de PWM, I2C, SPI, etc.
- Tensão de operação: 4,5 – 9V
- Taxa de transferência: 110-460800bps

- Suporta Upgrade remoto de *firmware*
- Conversor analógico digital (ADC)
- Distância entre pinos: 2,54mm
- Dimensões: 49 x 25,5 x 7 mm

Preço: R\$ 59,90

Fonte: ([FILIPEFLOP.COM](http://FILIPEFLOP.COM), 2017b)

- Sensor de Movimento Presença PIR DYP-ME003 ([Figura 38](#))

Figura 38 – Sensor de Movimento



Fonte: ([FILIPEFLOP.COM](http://FILIPEFLOP.COM), 2017c)

Especificações:

- Modelo: DYP-ME003
- Sensor Infravermelho com controle na placa
- Sensibilidade e tempo ajustável
- Tensão de Operação: 4,5-20V
- Tensão Dados: 3,3V (Alto) - 0V (Baixo)
- Distância detectável: 3-7m (Ajustável)
- Tempo de Delay: 5-200seg (*Default*: 5seg)
- Tempo de Bloqueio: 2,5seg (*Default*)
- *Trigger*: (L)-Não Repetível (H)-Repetível (*Default*: H)
- Temperatura de Trabalho: -20° a 80°C
- Dimensões: 3,2 x 2,4 x 1,8cm
- Peso: 7g

Preço: R\$ 12,90

Fonte: ([FILIPEFLOP.COM](http://FILIPEFLOP.COM), 2017c)

- Sensor de Temperatura LM35 ([Figura 39](#))

Figura 39 – Sensor de Temperatura



Fonte: ([FILIPEFLOP.COM](http://FILIPEFLOP.COM), 2017)

Especificações:

- Sensor de Temperatura LM35
- Faixa de temperatura: -0°C a 100°C
- Precisão: 0,5°C
- Calibrado em graus Celsius
- Tensão de operação: 4 a 30V
- Consumo de corrente: <60  $\mu$ A

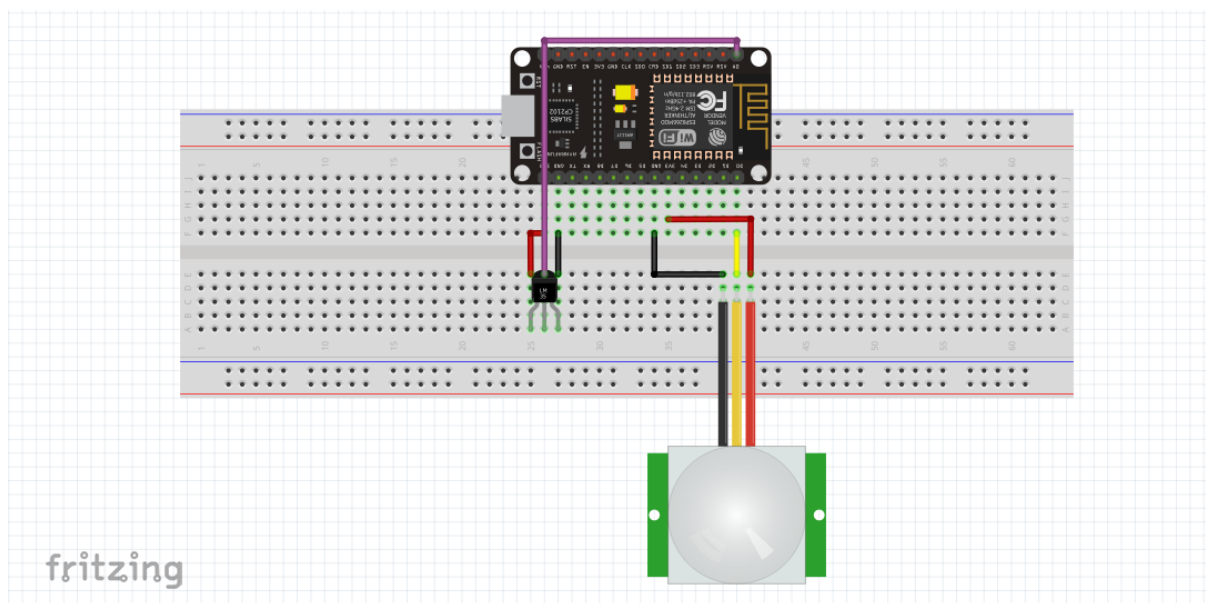
Preço: R\$ 8,90

Fonte: ([FILIPEFLOP.COM](http://FILIPEFLOP.COM), 2017)

Preço total dos materiais: R\$ 81,70

Na aplicação embarcada, foram utilizados uma placa NodeMCU, um sensor de temperatura e um sensor de movimento. A conexão dos sensores na placa foi feita em uma *Protoboard*. O esquema elétrico da conexão é mostrado na Figura 40.

Figura 40 – Esquema elétrico



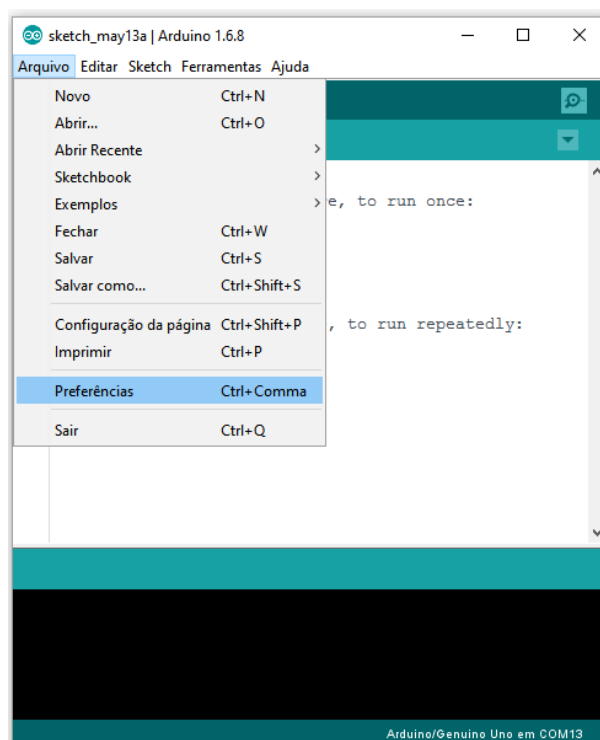
Fonte: Os autores, 2017

O NodeMCU é programado na linguagem Lua, mas também pode ser programado utilizando-se a linguagem padrão do Arduino, por meio do seu IDE. A versão utilizada foi a 1.8.0, disponível para *download* gratuito no endereço <https://www.arduino.cc/en/Main/Software>.

No entanto, antes de começar a programar, é preciso configurar o IDE para que ele reconheça o NodeMCU, adicionando-se a biblioteca responsável por gerenciar a placa ESP8266. Os procedimentos são descritos a seguir. Fonte: (FILIPEFLOP.COM, 2017a)

Entre no IDE do Arduino e clique em Arquivo -> Preferências (Figura 41):

Figura 41 – Menu Preferências



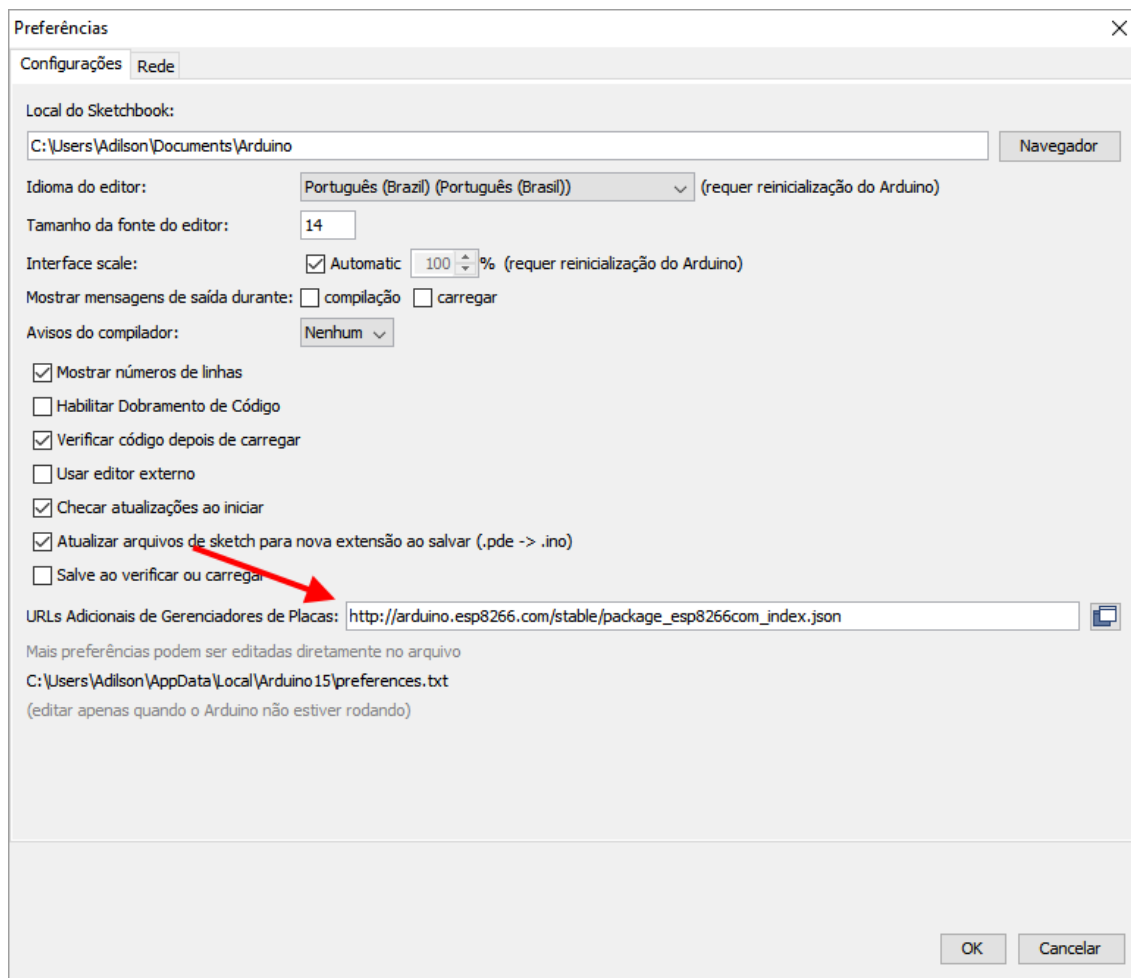
Fonte: ([FILIPEFLOP.COM](http://FILIPEFLOP.COM), 2017a)

Na tela seguinte, digite o *link* abaixo no campo URLs adicionais de Gerenciadores de Placas:

`http://arduino.esp8266.com/stable/package_esp8266com_index.json`

A sua tela ficará assim ([Figura 42](#)):

Figura 42 – Preferências



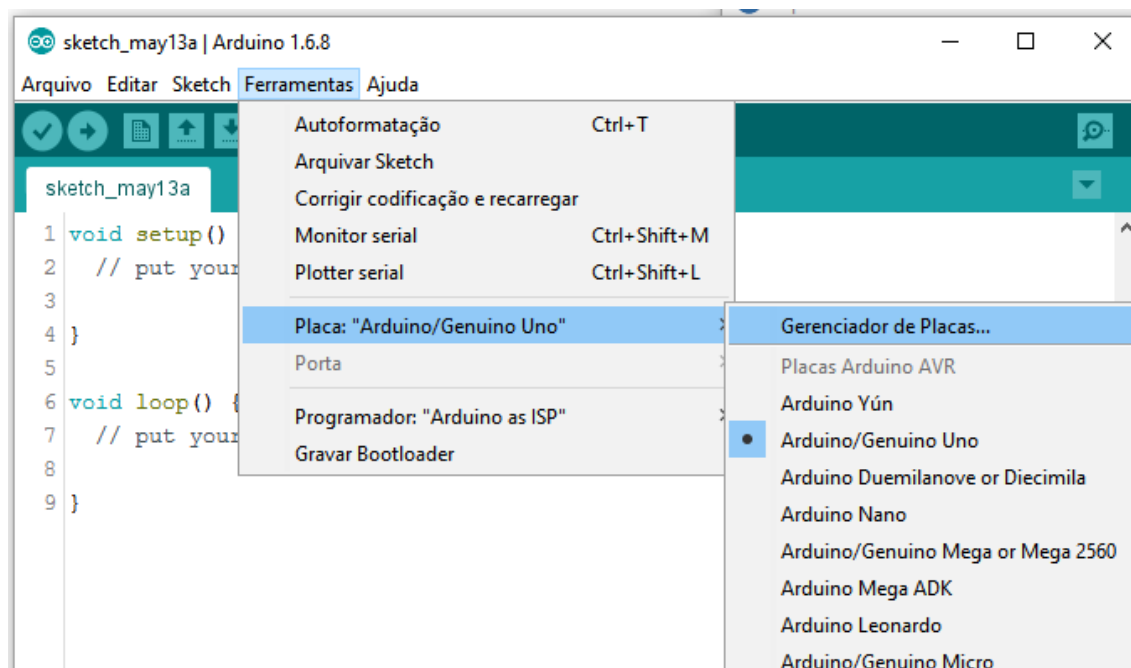
Fonte: (FILIPEFLOP.COM, 2017a)

Clique em OK para retornar à tela principal do IDE

Agora clique em Ferramentas -> Placa -> Gerenciador de Placas (Figura 43):



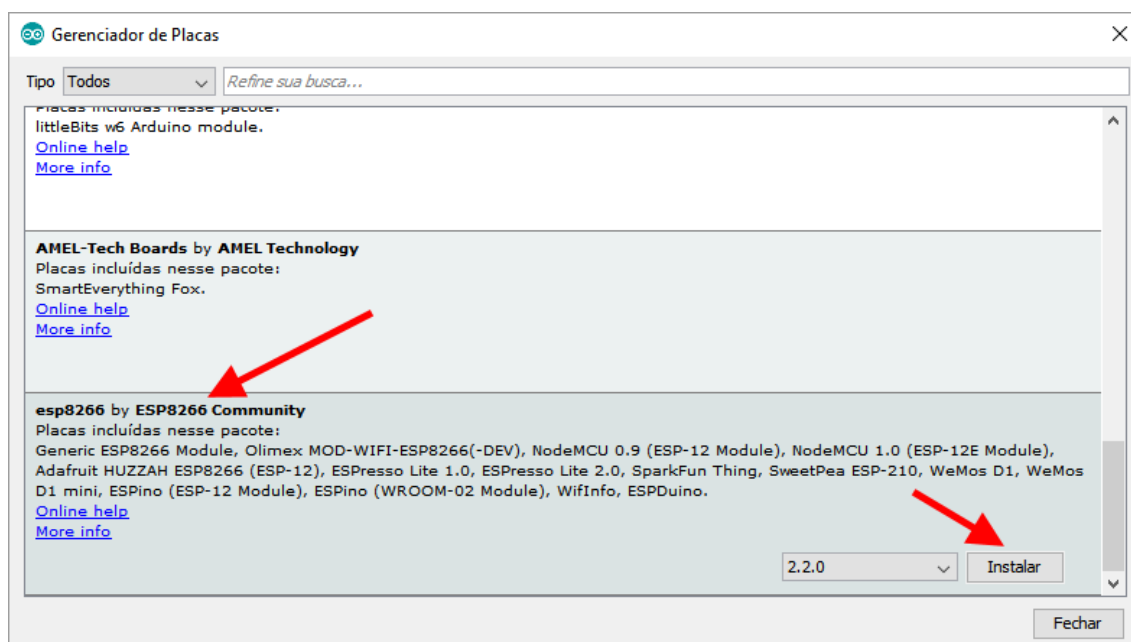
Figura 43 – Gerenciar Placas



Fonte: (FILIPEFLOP.COM, 2017a)

Utilize a barra de rolagem para encontrar o esp8266 by ESP8266 Community e clique em INSTALAR (Figura 44):

Figura 44 – Instalar biblioteca

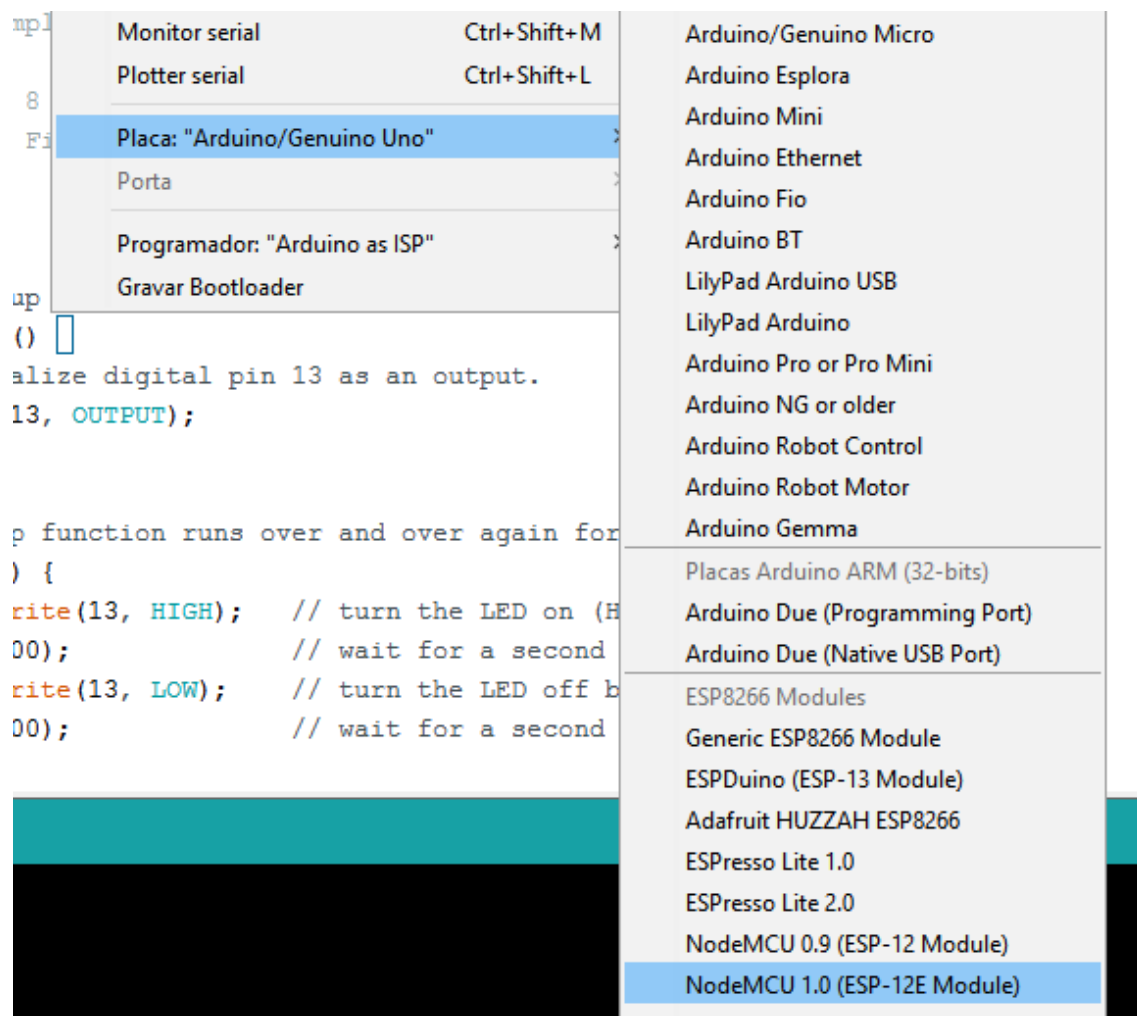


Fonte: (FILIPEFLOP.COM, 2017a)

Após alguns minutos as placas da linha ESP8266 já estarão disponíveis na lista de placas da IDE do Arduino.

No menu Ferramentas -> Placas, selecione a placa NodeMCU 1.0 (ESP 12-E module) (Figura 45):

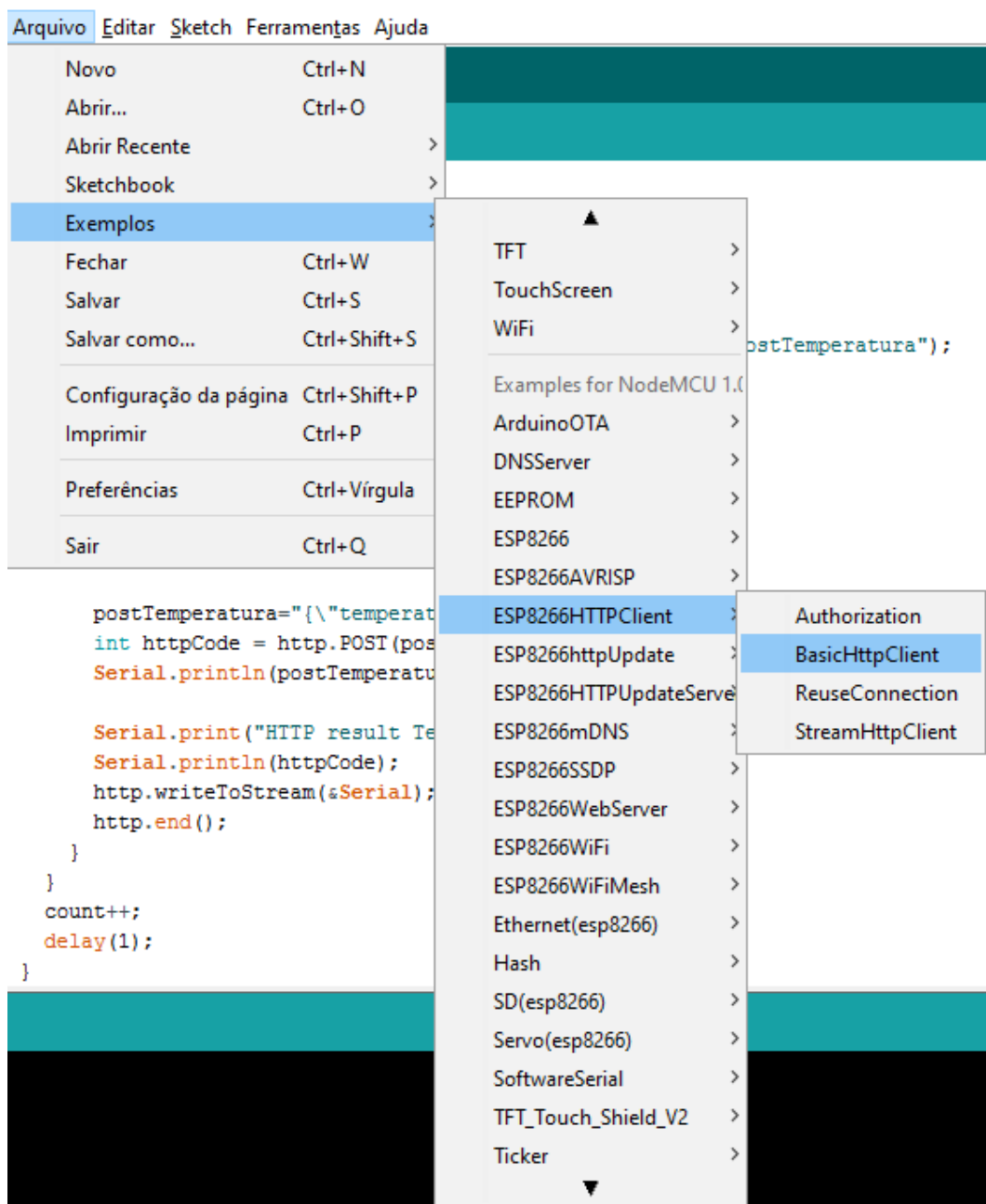
Figura 45 – Escolhendo a placa NodeMCU



Fonte: (FILIPEFLOP.COM, 2017a)

Para o código que roda no NodeMCU, foram utilizados alguns comandos presentes do exemplo de código "ESP8266HTTPClient: BasicHttpClient", disponível no IDE do Arduino, no menu Arquivo -> Exemplos -> ESP8266HTTPClient -> BasicHttpClient (Figura 46).

Figura 46 – Escolha do exemplo



Fonte: Os autores, 2017

Antes da função "*setup*", são incluídas as bibliotecas necessárias, declaradas as variáveis auxiliares e são nomeados os pinos do NodeMCU onde os sensores serão conectados, como mostrado na [Figura 47](#).

Figura 47 – Comandos prévios

```

#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266HTTPClient.h>
#define USE_SERIAL Serial

ESP8266WiFiMulti WiFiMulti;

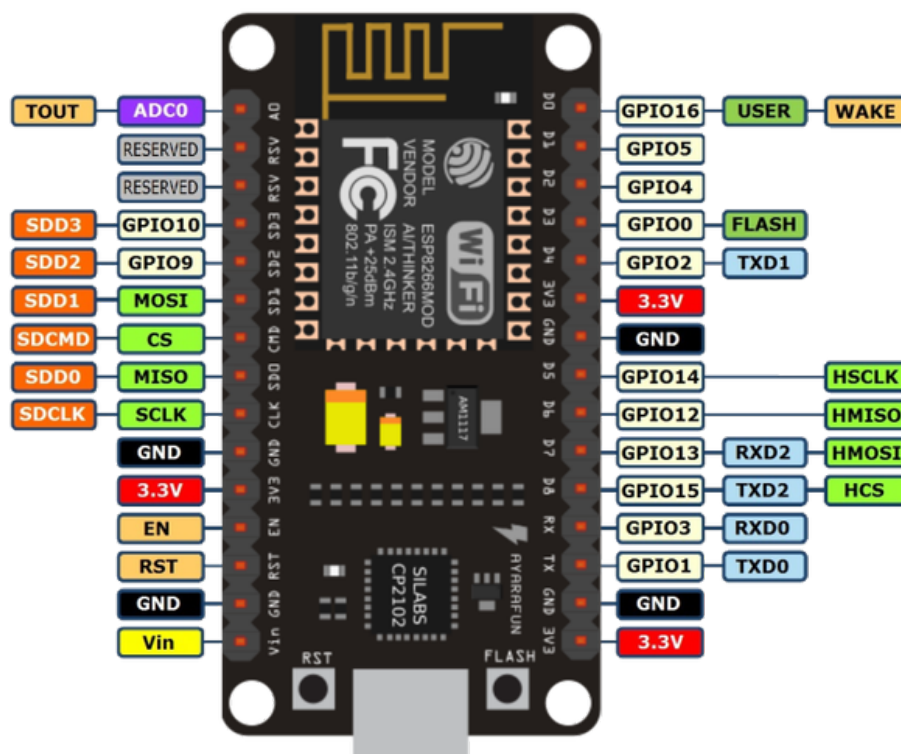
const int SensorTempPin = A0;
const int SensorMovPin = 16;
int count = 0;
int estado = 0;
String ip = "192.168.0.2";
//String ip = "jhs.hopto.org";

```

Fonte: Os autores, 2017

Os pinos onde os sensores foram conectados podem ser consultados no diagrama de pinos (*pinout diagram*) da placa (Figura 48).

Figura 48 – Diagrama de pinos



Fonte: (ROBOTS, 2017)

A variável auxiliar "ip", do tipo *string*, é utilizada ao longo do código nas URLs para as quais o NodeMCU deve enviar o comando POST. Para que não seja necessário

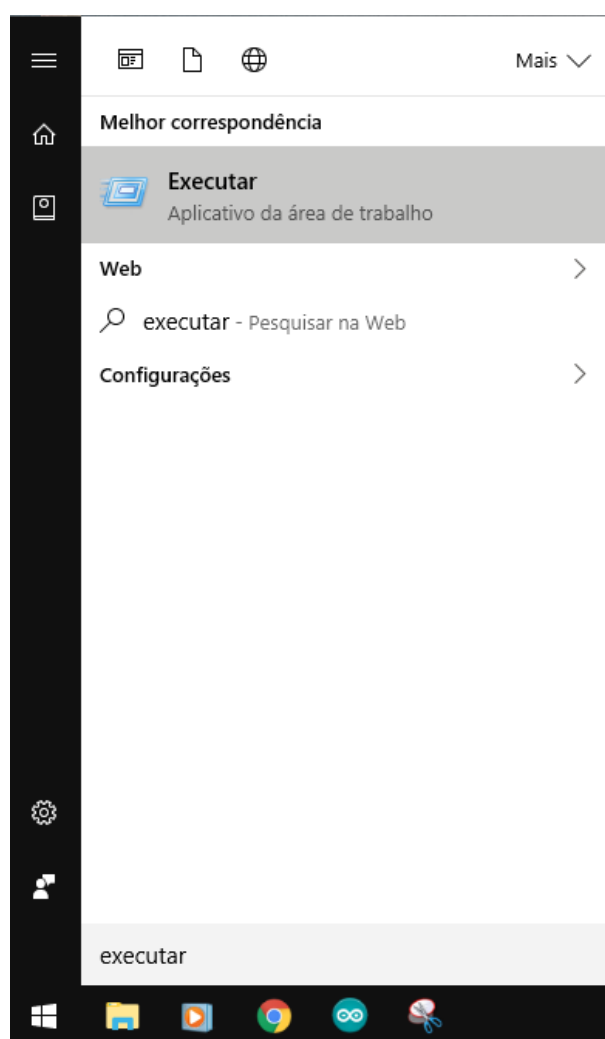
alterar em todos os locais do código cada vez que a rede é trocada, declaramos o IP na variável auxiliar, e no código, declaramos a URL concatenada com a variável, por exemplo:

```
"http://" + ip + ":8080/ServerREST/resources/postTemperatura"
```

Caso o Servidor REST esteja funcionando na sua rede local, o IP a ser utilizado pode ser encontrado fazendo-se o seguinte procedimento:

Abra o menu iniciar e pesquise pelo aplicativo "Executar" (Figura 49).

Figura 49 – Aplicativo Executar

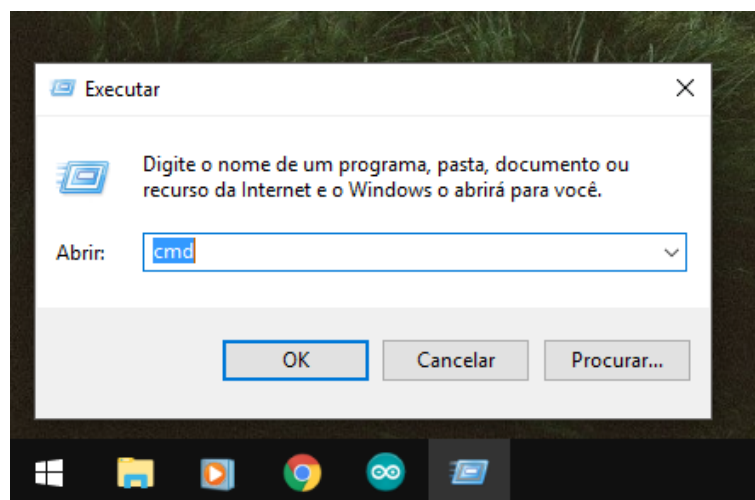


Fonte: Os autores, 2017

Obs.: Esse aplicativo também é aberto apertando as teclas Win+R (Figura 50).

Abra ele e pesquise por "cmd".

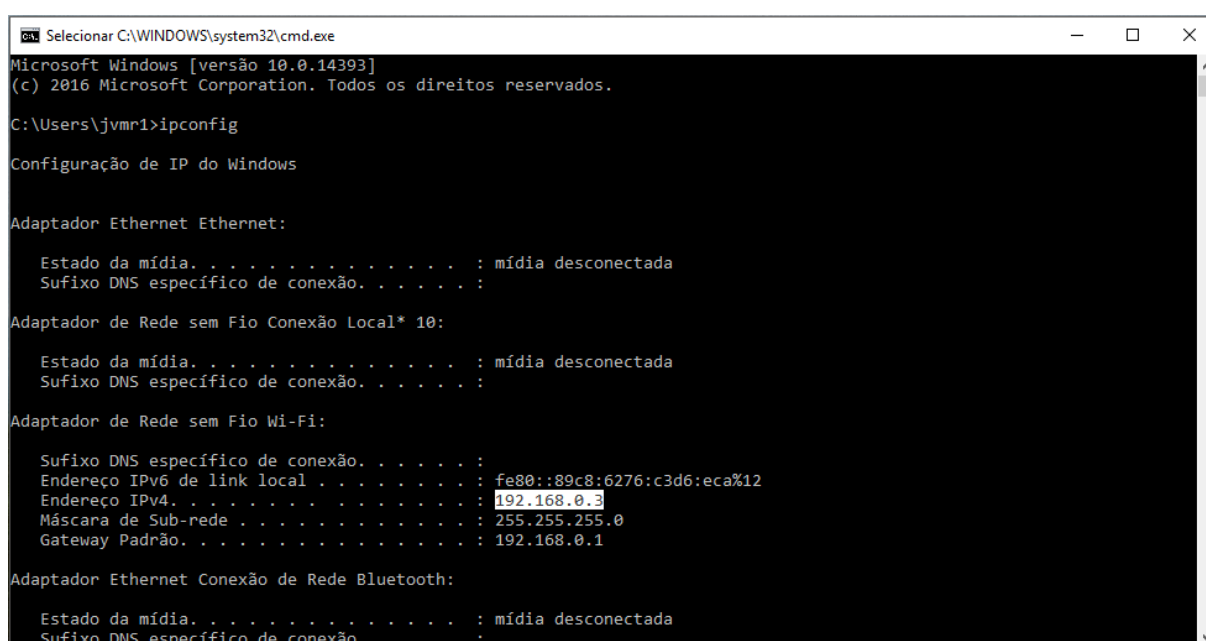
Figura 50 – Executar comando cmd



Fonte: Os autores, 2017

O Prompt de comando será aberto. Digite "ipconfig" e dê enter. O IP a ser utilizado na variável "ip" do código estará indicado como "Endereço IPv4" (Figura 51).

Figura 51 – Comando ipconfig no Prompt



Fonte: Os autores, 2017

Obs.: Caso o programa deva enviar as mensagens POST para um endereço na internet, pode-se colocar a URL do site no local do IP. Exemplo:

```
String ip = "jhs.hopto.org";
```

Na função "setup", no comando:

```
WiFiMulti.addAP("SSID", "PASSWORD");
```

Define-se o nome e a senha da rede *WiFi* que o módulo deverá se conectar, trocando a palavra "SSID" pelo nome da rede, e "PASSWORD" pela senha, mantendo as aspas em ambos os casos. Exemplo:

```
WiFiMulti.addAP("JH", "12345678");
```

É recomendado conectar na mesma rede *WiFi* que o computador está conectado. Ainda na função "setup", definem-se os modos dos pinos (INPUT ou OUTPUT). No caso de sensores, onde haverá entrada de dados, escolhe-se o modo INPUT. Caso haja um LED (*Light Emitting Diode*) ou algum outro elemento de saída de dados, seu modo é posto como OUTPUT, como exibido na Figura 52.

Figura 52 – Função *setup*

```
void setup() {  
    USE_SERIAL.begin(115200);  
    USE_SERIAL.println();  
    USE_SERIAL.println();  
    USE_SERIAL.println();  
  
    for (uint8_t t = 4; t > 0; t--) {  
        USE_SERIAL.printf("[SETUP] WAIT %d...\n", t);  
        USE_SERIAL.flush();  
        delay(1000);  
    }  
  
    WiFiMulti.addAP("JH", "12345678");  
  
    pinMode(SensorTempPin, INPUT);  
    pinMode(SensorMovPin, INPUT);  
    Serial.begin(115200);  
}
```

Fonte: Os autores, 2017

Na função "loop", é colocado o algoritmo que será executado. No presente experimento, a seguinte lógica foi empregada: A medição da temperatura é feita a cada intervalo de tempo predefinido, e a cada medição, a mensagem é enviada para o servidor pelo método POST. O sensor de movimento, ao ser ativado, envia instantaneamente uma mensagem para o servidor pelo mesmo método. O algoritmo completo pode ser encontrado no final deste Apêndice.

Substitui-se o endereço no comando "http.begin" pelo endereço para o qual o programa enviará mensagens pelo método POST. A URL é encontrada manualmente no NetBeans. Exemplos:

```
http.begin("http://" + ip + ":8080/ServerREST/resources/postPresenca");
```

```
http.begin("http://" + ip + ":8080/ServerREST/resources/postTemperatura");
```

Observa-se que os serviços responsáveis por cada medição (do sensor de temperatura e do de movimento) são independentes.

Feitas todas estas modificações, o programa está pronto. Ao ser enviado para a

placa, o processo de conexão do módulo é iniciado, os dados dos sensores são lidos e enviados para o servidor indicado por meio do método POST.

## B.1 Código do Programa

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266WiFiMulti.h>
#include <ESP8266HTTPClient.h>
#define USE_SERIAL Serial

ESP8266WiFiMulti WiFiMulti;

const int SensorTempPin = A0;
const int SensorMovPin = 16;
int count = 0;
int estado = 0;
String ip = "jhs.hopto.org";
//String ip = "192.168.0.2";

void setup() {
    USE_SERIAL.begin(115200);
    USE_SERIAL.println();
    USE_SERIAL.println();
    USE_SERIAL.println();

    for (uint8_t t = 4; t > 0; t--) {
        USE_SERIAL.printf("[SETUP] WAIT %d...\n", t);
        USE_SERIAL.flush();
        delay(1000);
    }

    WiFiMulti.addAP("JH", "12345678");

    pinMode(SensorTempPin, INPUT);
    pinMode(SensorMovPin, INPUT);
    Serial.begin(115200);
}
```



```
void loop() {
    if ((WiFiMulti.run() == WL_CONNECTED)) {
        //Sensor de movimento
        bool movimento;
        movimento=digitalRead(SensorMovPin);

        if (movimento == HIGH && estado == 0) {
            estado = 1;
            Serial.println("{\"sensornumero\":\"1\"}");
            HTTPClient http;
            http.begin("http://" + ip + ":8080/ServerREST/resources/postPresenca");
            http.addHeader("Content-Type", "text/json");

            movimento = digitalRead(SensorMovPin);
            String postMessage = "{\"sensornumero\":\"1\"}";
            int httpCode = http.POST(postMessage);

            Serial.print("HTTP result Sensor: ");
            Serial.println(httpCode);
            http.writeToStream(&Serial);
            http.end();
        }

        if (movimento == LOW) {
            estado = 0;
        }

        //Sensor de temperatura
        if (count > 10000) {
            count = 0;
            HTTPClient http;
            http.begin("http://" + ip + ":8080/ServerREST/resources/postTemperatura");
            http.addHeader("Content-Type", "text/json");

            String postTemperatura;
            int leitura;
            float temperatura;
            leitura = analogRead(SensorTempPin);
```

```
temperatura = ( 3.3 * leitura * 100.0) / 1023.0;

postTemperatura="{\"temperatura\":\":" + String(temperatura) + "\",\"
int httpCode = http.POST(postTemperatura);
Serial.println(postTemperatura);

Serial.print("HTTP result Temperatura: ");
Serial.println(httpCode);
http.writeToStream(&Serial);
http.end();
}
}
count++;
delay(1);
}
```

# APÊNDICE C – Criando aplicação consumidor do serviço no Android

A criação de aplicativos para o sistema Android é feita principalmente pelo IDE do Android Studio, uma ferramenta desenvolvida e disponibilizada pela própria Google, detentora da marca Android. A linguagem de programação utilizada no desenvolvimento foi Java, e para a criação de interfaces, o XML.

O aplicativo criado, de nome "consumidor REST", foi feito na versão 2.2 do Android Studio. O processo de criação de um novo projeto é simples, na tela principal, clicar em File -> Project -> New Project, e então dar o nome e escolher o layout da tela inicial.

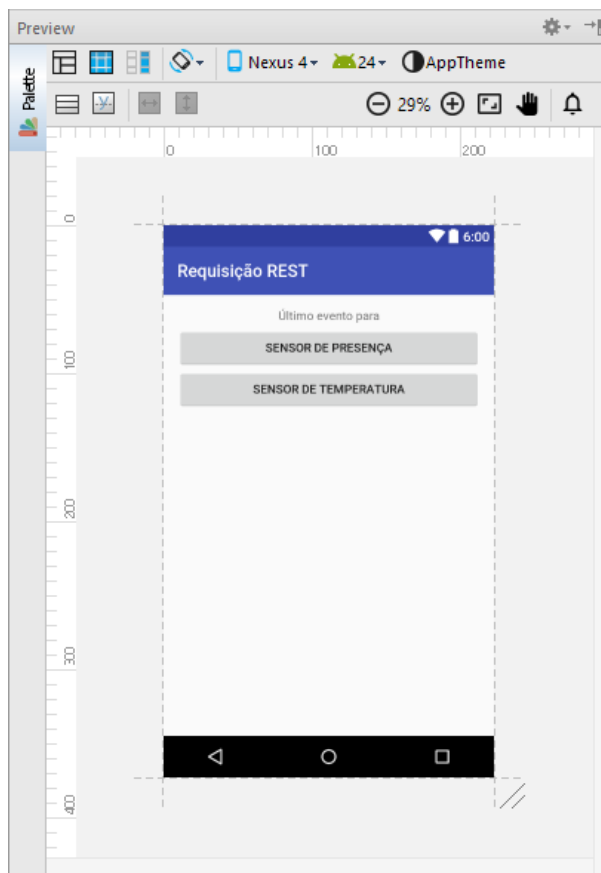
## C.1 Definindo o layout de apresentação e funcionalidades

O primeiro passo do projeto, nesse caso, foi definir a sua apresentação. O aplicativo é composto apenas de uma tela com dois botões que solicitam, cada um, um serviço do servidor. No caso, um botão solicita dados do sensor de temperatura e o outro do sensor presença. Ao solicitar a informação, os dados do último evento dos sensores deve ser mostrados na tela, para isso, foram criados abaixo dos botões, oito visualizadores de texto invisíveis, que são apresentados de acordo com as solicitações.

A o apertar o botão de informação do sensor de presença, seis visualizadores de texto se tornam visíveis, mostrando as informações obtidas. Para o botão de solicitação de informação de temperatura, oitos visualizadores se tornam visíveis.

Parte do código XML de interface e o design final, que é feita na classe "Activity Principal", é apresentado na [Figura 53](#).

Figura 53 – Layout XML do aplicativo

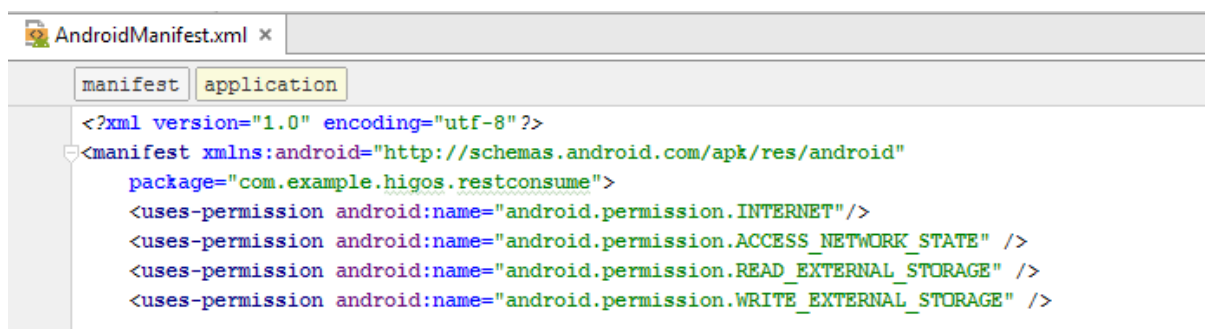


Fonte: Os autores, 2017

## C.2 Conectando o aplicativo ao servidor

Para ser possível uma conexão com uma rede local ou na internet, é preciso solicitar permissão no Android, na classe `AndroidManifest`, para ter acesso a seus recursos de rede. Os comando de permissão são mostrados na [Figura 54](#).

Figura 54 – Permissões de acesso



Fonte: Os autores, 2017

Toda solicitação de rede feita pelo aplicativo deve ser feita com o uso de *threads*

para que não trave a interface, portanto, para o processo, é preciso herdar uma classe chamada `AsyncTask`, que faz a execução do seu código-fonte em um outro núcleo do processador. Dentro dessa classe, foi feita a chamada `GET` para o servidor, passando o parâmetro `URL` do serviços solicitado. O código responsável por essa função é encontrado na [Figura 55](#).

Figura 55 – Classe de conexão

```
public class JSONAsyncTask extends AsyncTask <URL, Integer, StringBuffer> {  
    protected StringBuffer doInBackground(URL... urls) {  
        StringBuffer chaine = new StringBuffer("");  
        try {  
            System.out.println(urls[0]);  
            HttpURLConnection connection = (HttpURLConnection)  
                urls[0].openConnection();  
            connection.setRequestMethod("GET");  
            connection.connect();  
            System.out.println("Responde Code: " + connection.getResponseCode());  
            InputStream inputStream = connection.getInputStream();  
            BufferedReader rd = new BufferedReader(new InputStreamReader(inputStream));  
            String line = "";  
            while ((line = rd.readLine()) != null) {  
                chaine.append(line);  
            }  
            System.out.println("chaine: " + chaine);  
  
            return chaine;  
        } catch (IOException e) {  
            System.out.println("falhou miseravelmente");  
            e.printStackTrace();  
            return null;  
        }  
    }  
}
```

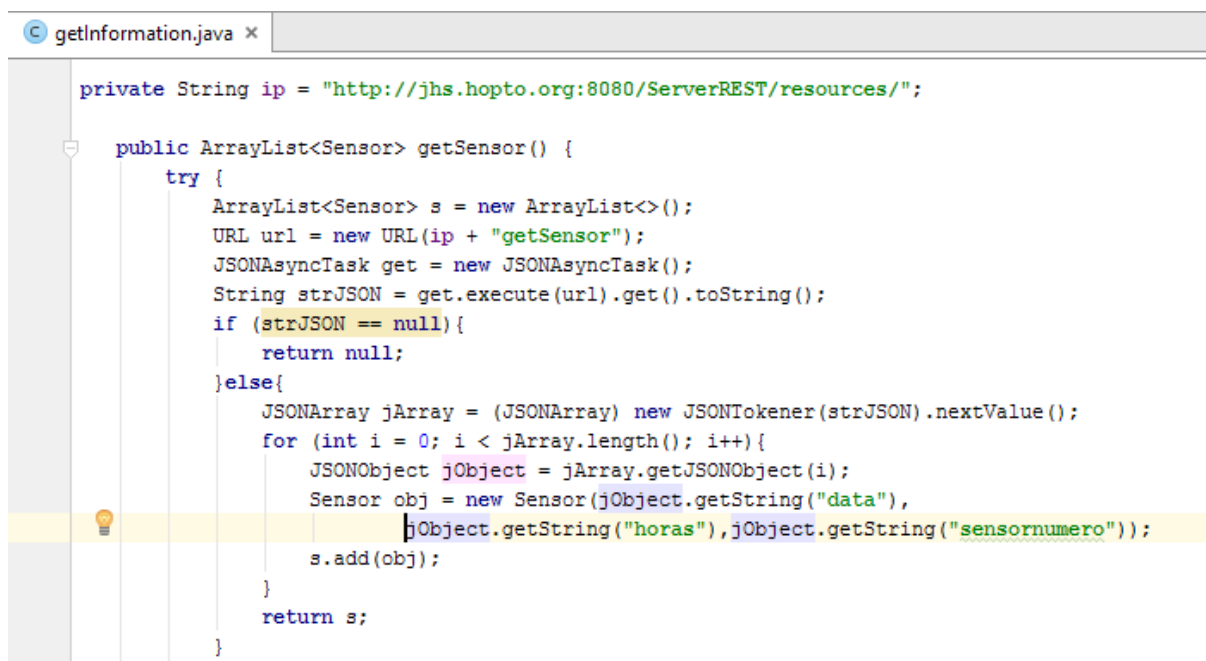
Fonte: Os autores, 2017

Ao ser executada, essa classe retorna um dado do tipo `StringBuffer` com a resposta recebida do servidor.

### C.3 Trabalhando com JSON

Para a conversão dos eventos recebidos em JSON para objetos manipuláveis no Java, foi utilizada a classe `JSONArray`, que transforma suas informações em um vetor, logo em seguida, esse vetor é percorrido, transformando seus dados em vetores de objetos do seu tipo, conforme mostra a [Figura 56](#).

Figura 56 – Conversão do JSON



```
private String ip = "http://jhs.hopto.org:8080/ServerREST/resources/";

public ArrayList<Sensor> getSensor() {
    try {
        ArrayList<Sensor> s = new ArrayList<>();
        URL url = new URL(ip + "getSensor");
        JSONAsyncTask get = new JSONAsyncTask();
        String strJSON = get.execute(url).get().toString();
        if (strJSON == null) {
            return null;
        } else {
            JSONArray jArray = (JSONArray) new JSONTokener(strJSON).nextValue();
            for (int i = 0; i < jArray.length(); i++) {
                JSONObject jObject = jArray.getJSONObject(i);
                Sensor obj = new Sensor(jObject.getString("data"),
                    jObject.getString("horas"), jObject.getString("sensornumero"));
                s.add(obj);
            }
            return s;
        }
    }
}
```

Fonte: Os autores, 2017

Foi criado um método para cada serviço que, ao ser chamado, retorna um vetor com os objetos de todos os eventos recebidos do servidor.

## C.4 Chamando os recursos

Na classe principal é feita a interação entre a interface e os processos, é criado um evento de escuta para os botões, onde ao ser clicado, ele coloca o último evento do vetor para exibir nos visualizadores de texto, transformando eles em visíveis na tela, como demonstra a [Figura 57](#).

Figura 57 – Classe principal

```
Button sensor = (Button) findViewById(R.id.botaosensor);

sensor.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        getInformation g = new getInformation();
        ArrayList<Sensor> a = g.getSensor();

        numsensor.setText("Sensor " + a.get(a.size() - 1).getSensornumero());
        data.setText(a.get(a.size() - 1).getData());
        hora.setText(a.get(a.size() - 1).getHoras());

        texto.setVisibility(View.VISIBLE);
        numsensor.setVisibility(View.VISIBLE);
        texto1.setVisibility(View.VISIBLE);
        data.setVisibility(View.VISIBLE);
        texto2.setVisibility(View.VISIBLE);
        hora.setVisibility(View.VISIBLE);
        texto3.setVisibility(View.INVISIBLE);
        temperatura.setVisibility(View.INVISIBLE);
    }
}
```

Fonte: Os autores, 2017