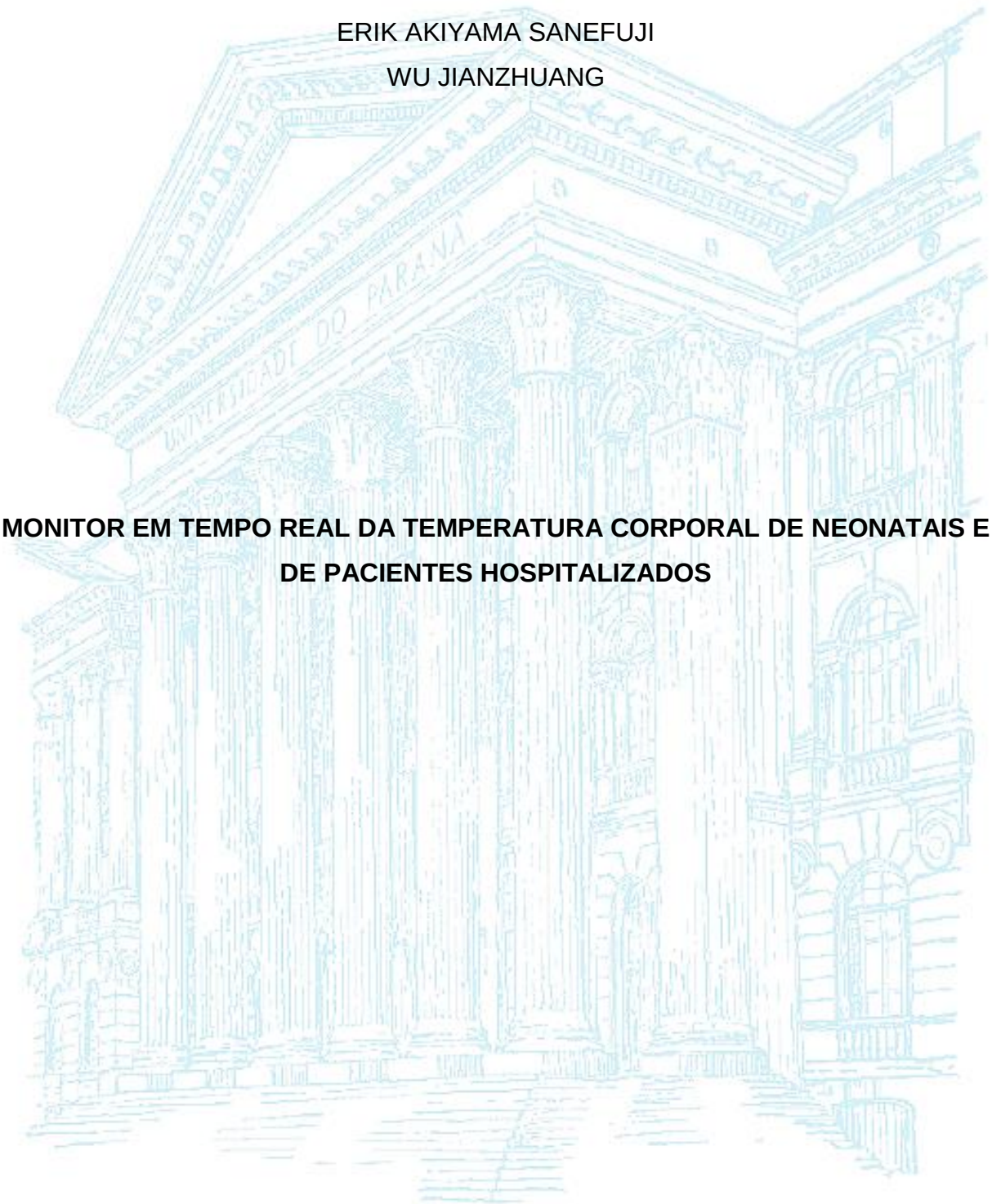
UNIVERSIDADE FEDERAL DO PARANÁ
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

ERIK AKIYAMA SANEFUJI
WU JIANZHUANG



MONITOR EM TEMPO REAL DA TEMPERATURA CORPORAL DE NEONATAIS E DE PACIENTES HOSPITALIZADOS

CURITIBA
2018

ERIK AKIYAMA SANEFUJI
WU JIANZHUANG

MONITOR EM TEMPO REAL DA TEMPERATURA CORPORAL DE NEONATAIS E DE PACIENTES HOSPITALIZADOS

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do grau de Engenheiro Eletricista com ênfase em Sistemas Eletrônicos Embarcados da Universidade Federal do Paraná

Orientador: Prof. Dr. Marcos Vinicio Haas Rambo

CURITIBA
2018

TERMO DE APROVAÇÃO

ERIK AKIYAMA SANEFUJI
WU JIANZHUANG

MONITOR EM TEMPO REAL DA TEMPERATURA CORPORAL DE NEONATAIS E DE PACIENTES HOSPITALIZADOS

Trabalho de conclusão de curso apresentado como requisito parcial para à obtenção do grau de Engenheiro Eletricista com Ênfase em Sistemas Eletrônicos Embarcados, do Setor de Tecnologia da Universidade Federal do Paraná, pela seguinte banca examinadora:

_____
Prof. Dr. Marcos Vinicio Haas Rambo
Departamento de Engenharia Elétrica, UFPR

_____
Prof. Dr. Armando Heilmann
Departamento de Engenharia Elétrica, UFPR

_____
Prof. Rafael da Silva Ferraz
Departamento de Engenharia Elétrica, UFPR

Curitiba, 4 de dezembro de 2018.

Dedicamos este trabalho ao nosso orientador, por seus ensinamentos, paciência e confiança ao longo deste trabalho, e a nós mesmo pelo esforço realizado em conjunto e a coragem para tentar resolver desafios complexos em nossa sociedade.

AGRADECIMENTOS

Agradecemos primeiramente às nossas famílias, que nos suportaram arduamente ao longo da execução deste trabalho, e também nos apoiaram em momentos de cansaço e frustração durante toda a graduação. Agradecemos também a todos os professores, que nos passaram o conhecimento necessário para compreender melhor o Universo que nos cerca e como modificá-lo para o bem da sociedade. Em especial gostaríamos de agradecer ao professor Dr. Marcos Vinicio Haas Rambo, nosso orientador, pelo esforço dedicado em nos guiar não apenas neste projeto, mas durante toda a graduação, como referência profissional e pessoal, e ao professor Dr. João da Silva Dias, pelas incontáveis horas corrigindo nossa redação e fornecendo ideias para o projeto.

LISTA DE FIGURAS

LISTA DE ANEXOS

LISTA DE TABELAS

LISTA DE SIGLAS

6LoWPAN - *IPv6 over Low Power Wireless Personal Area Networks*
ABNT - Associação Brasileira de Normas Técnicas
ADC - *Analog to Digital Converter*
Anvisa - Agência Nacional de Vigilância Sanitária
API - *Application Programming Interface*
APP - Associação Brasileira de Pediatria
ARM - *Acorn RISC Machine*
ARPA - *Advanced Research Projects Agency*
BLE - *Bluetooth Low Energy*
BR - *Bluetooth Ready*
CATNIP - *Common Architecture for the Internet*
D2D - *Device-to-Device*
DAC - *Digital to Analog Converter*
DHCP - *Dynamic Host Control Protocol*
DoD - *Department of Defense*
DSP - *Digital Signal Processor*
EDR - *Enhanced Data Rate*
EEPROM - *Electrically Erasable Programmable Read-Only Memory*
FPGA - *Field Programmable Gate Arrays*
GNU - *GNU is Not Unix*
HTTP - *HyperText Transfer Protocol*
HTTPS - *HyperText Transfer Protocol Secured*
I2C - *Inter-Integrated Circuit*
IANA - *Internet Assigned Numbers Authority*
IEEE - *Institute of Electrical and Electronics Engineers*
IETF - *Internet Engineering Task Force*
IoT - *Internet of Things*
IP - *Internet Protocol*
IPAE - *IP Address Encapsulation*
IPng - *Internet Procotol next Generation*
IPv4 - *Internet Procotol version 4*
IPv6 - *Internet Procotol version 6*

IrDA - *Infrared Data Association*
M2M - *Machine-to-Machine*
MCU - *Main Control Unit*
NAT - *Network Address Translation*
NCP - *Network Control Protocol*
NCSA - *National Center for Supercomputing Applications*
NFC - *Nearby Field Communication*
P2P - *Peer-to-Peer*
PIP - *Paul's Internet Protocol*
QoS - *Quality of Service*
RAM - *Random Access Memory*
RFC - *Request for Comments*
ROM - *Read Only Memory*
SD - *Secure Disk*
SIP - *Simple Internet Protocol*
SIPP - *Simple Internet Protocol Plus*
SoC - *System-on-Chip*
SPI - *Serial Peripheral Interface*
TCP - *Transmission Control Protocol*
TUBA - *TCP and UDP with Bigger Addresses*
UART - *Universal Asynchronous Receiver Transmitter*
UDP - *User Datagram Protocol*
USART - *Universal Synchronous Asynchronous Receiver Transmitter*
VoIP - *Voice over IP*
VPN - *Virtual Private Network*
WAN - *Wide Area Network*

1. INTRODUÇÃO

1.1. MOTIVAÇÃO

O cuidado de crianças recém-nascidas requer muita atenção pelos pais e pediatras, pois a sua saúde ainda é muito frágil. Frequentemente elas ficam doentes e a temperatura corporal é o principal parâmetro indicativo da presença de inflamações ou infecções. Quando estes problemas são identificados, eles devem ser tratados rapidamente para que eles não prejudiquem a saúde e o desenvolvimento da criança (healthychildren.org).

Durante o tratamento, a medição da temperatura vira uma rotina para os pais e deve ser monitorada com uma frequência bem definida, para avaliar a eficácia do tratamento e a melhora da criança. Os períodos noturnos são sempre os mais desgastantes. Ainda, os pais têm a necessidade de garantir o sustento do lar, e situações em que eles deixam o bebê em creches ou em casa com cuidadoras, por causa do trabalho, são muito comuns.

Tudo isso acaba dificultando o monitoramento da temperatura do bebê pelos pais. Sem contar que, ao chegar ao consultório médico ou hospital, o profissional de saúde possa querer saber como estava o comportamento da febre da criança para poder melhor avaliar a saúde da mesma e a eficácia do tratamento, pois a febre é a resposta do corpo a um problema. Ela pode ser o resultado de uma simples inflamação ou uma doença grave. Através do comportamento da temperatura, é possível saber se a criança está saudável e se o tratamento está respondendo da forma esperada, sem mencionar que muitas crianças entram em convulsão devido às temperaturas corporais elevadas.

O monitoramento *online* facilitaria a atuação imediata dos pais em tal situação. Em quantos graus estava a febre da criança pela manhã, ao meio dia, à tarde ou à noite, por exemplo, são dados importantes. Muito provavelmente os pais e os pediatras gostariam de ter esta informação em forma de histórico e com geração de alarmes, quando a temperatura corporal do bebê ultrapassar limites pré-estabelecidos.

Outra situação crítica, no que diz respeito ao monitoramento da temperatura corporal, é a de um paciente internado em um hospital. Este monitoramento é de extrema importância para o acompanhamento do tratamento e da sua recuperação.

A medição deve ser feita periodicamente. Isto gera um trabalho manual e repetitivo, contribuindo com a sobrecarga de trabalho dos enfermeiros e auxiliares de enfermagem, que devem atualizar o prontuário médico a cada medição de temperatura.

Se a temperatura do paciente pudesse ser obtida facilmente através de um bracelete e integrada a sistemas inteligentes em aparelhos celulares, *tablets* ou computadores, não haveria mais a necessidade da medição manual. As informações de temperatura poderiam facilmente integrar o prontuário médico eletrônico do paciente, de forma autônoma, reduzindo a carga de trabalho dos enfermeiros e aumentando a confiabilidade das informações fornecidas ao médico. Isto também facilitaria o acesso a partir de qualquer lugar, pois as informações estariam facilmente na *internet*.

Por esses motivos, neste presente trabalho, optou-se por desenvolver um bracelete para monitoramento *online* da temperatura de bebês e de pacientes hospitalizados integrados à internet, usando o conceito de internet das coisas (*Internet of Things* – IoT), com geração de alarmes e de dados históricos. O sistema a ser desenvolvido também será capaz de enviar uma mensagem alertando o médico ou os pais sobre a febre da criança ou do paciente.

Mas para identificar se o paciente está ou não com febre, é essencial conhecer a temperatura normal do corpo, conhecendo as condições médicas do paciente em questão. É importante lembrar que definir uma temperatura 'normal' do corpo não é simples, pois é um parâmetro variável. De acordo com Associação de Pediatria Americana (APP), “Os bebês tendem a ter temperaturas mais altas do que as crianças mais velhas, e a temperatura do todo é maior entre o final da tarde e início da noite e menor entre meia-noite e de manhã cedo”. Além disso, existem várias maneiras de medir a temperatura corporal. Pode ser via retal, via oral, axilar, na testa ou através da orelha. Portanto a temperatura corporal vai depender de onde medir, como medir, e o período do dia em que é medida. A temperatura dentro dos tecidos do corpo varia com a taxa metabólica e não há uma temperatura central bem estabelecida. Existe apenas uma faixa de temperatura aceita pela comunidade científica e médica. A qual será abordada com mais detalhes posteriormente. Estas questões serão avaliadas durante a etapa de revisão bibliográfica para poder identificar a melhor forma de medição e de como será feita a configuração do sistema e a geração de alarmes.

1.2. OBJETIVOS

1.2.1. Objetivo Geral

Desenvolvimento de um bracelete para monitoramento *online* da temperatura de bebês e de pacientes hospitalizados.

1.2.2. Objetivos específicos

- Enumerar produtos com proposta similar no mercado atual;
- Contrastar microcontroladores para o projeto;
- Desenvolver *firmware* do sistema;
- Avaliar sensores de temperatura;
- Avaliar ambientes em nuvem;
- Desenvolver aplicação,

1.3. ESTRUTURA DO TRABALHO

Com os assuntos apresentados, brevemente, este capítulo mostra a motivação do projeto a ser desenvolvido. No capítulo 2 é feita uma revisão bibliográfica dos temas a serem abordados durante o projeto, são estes: monitoramento em tempo real da temperatura de bebês e pacientes hospitalizados, *Internet of Things*, protocolos de comunicação sem fio e *system-on-chip.* Já no capítulo 3 é apresentado o desenvolvimento do projeto, como incluindo o diagrama final do bracelete e o diagrama em blocos. O capítulo 4 apresenta os resultados e discussões. O capítulo 5, trabalhos futuros e, por fim, o capítulo 6 conclui o desenvolvimento do projeto, explicitando as dificuldades e soluções utilizadas pelos integrantes de forma a facilitar a pesquisa.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. AQUISIÇÃO EM TEMPO REAL DA TEMPERATURA

O primeiro termômetro foi inventado por Galileu Galilei, um físico, matemático, astrônomo e filósofo. O termômetro dele consiste numa coluna de vidro cheia de um líquido onde se encontram imersos pequenos globos de vidro cheios do mesmo líquido. Cada bolha tem uma pequena etiqueta de metal que indica a temperatura. A densidade efetiva de cada globo é ajustada usando diferentes quantidades de líquido. Deste modo, quando a temperatura ambiente é superior a um dado valor, apresentado numa pequena placa presa ao globo, este flutua no cimo da coluna, caso contrário desce até o fundo da coluna. Portanto pode-se estimar a temperatura ambiente verificando qual a temperatura máxima indicada pelos globos que flutuam junto ao cimo da coluna (ACTFORLIBRARIES, 2016).

No século 16/17, um médico paracelsista, o alquimista Robert Fludd, inventou um equipamento conhecido como a termocópia. Embora o invento também tenha sido atribuído a Galileu, Santorio e Cornelius Drebbel, este dispositivo é conhecido por nós hoje como um termômetro de ar. Quando a extremidade aberta de um tubo fechado é submersa em água, alterações na temperatura do ar fazem com que a água no tubo suba ou desça, o que também faz com que este dispositivo seja susceptível à pressão do ar, funcionando também como um barômetro (ACTFORLIBRARIES, 2016).

A palavra termômetro é de origem francesa e é conhecida por ter sido usada pela primeira vez em 1624. O primeiro termômetro moderno, feito com fins completamente fechados e com conteúdo de álcool, foi feito cerca de 20 anos mais tarde pelo Grão-Duque da Toscana de Médici (ACTFORLIBRARIES, 2016).

A medição da temperatura do corpo, naquele momento na história do termômetro, levaria 20 minutos. Em 1866, Sir Thomas Allbutt criou um termômetro clínico que poderia tomar uma leitura de temperatura em 5 minutos (ACTFORLIBRARIES, 2016).

A escala do termômetro foi debatida até que Daniel Fahrenheit desenvolveu a escala Fahrenheit quase 100 anos após o termômetro de ar ter sido inventado. Essa escala foi desenvolvida utilizando mercúrio como o líquido no tubo de vidro, o que viria a se tornar o padrão até os anos 1990, quando os termômetros digitais começaram a substituir termômetros tradicionais na medicina, principalmente devido às

preocupações de segurança (ACTFORLIBRARIES, 2016).

Outro tipo de termômetro de invenção recente, usado para medir a temperatura do corpo, é chamado um termômetro basal. Estes dispositivos são usados para determinar a fertilidade e ovulação em mulheres que tentam engravidar. A maioria dos termômetros digitais são sensíveis o suficiente para medir também a temperatura basal (ACTFORLIBRARIES, 2016).

2.1.1. Principais tipos de termômetro

Existem vários tipos de termômetros no mercado. Eles podem ser classificados em dois principais grupos: termômetros de fluído e eletrônicos. Os de fluídos são os mais antigos e estão deixando de ser fabricados e comercializados para aplicações na área da saúde por questões de segurança. Já os termômetros do segundo grupo, além de modernos e precisos, são muito mais seguros e confiáveis.

a) Termômetros de mercúrio

São construídos em formato de um pequeno tubo fechado, com uma extremidade metálica, e seu interior contém mercúrio.

O mercúrio, como todos os materiais, dilata-se quando aumenta a temperatura. Ele é extremamente sensível, ou seja, altera seu volume à menor variação de temperatura, mesmo próxima à do corpo humano. O volume do mercúrio aquecido se expande no tubo capilar do termômetro.

Essa expansão é medida pela variação proporcional do comprimento, numa escala graduada que pode ter uma precisão de 0,05°C. É dessa forma, pela expansão do líquido, que é observada a variação da temperatura (SECRETARIA DO EDUCAÇÃO DO PARANÁ). Eles são os modelos mais antigos para medir a temperatura corporal, mas eles estão sendo descontinuados comercialmente.

A quebra acidental do termômetro provoca a exposição do líquido e fumos do seu interior, e estes são extremamente tóxicos para os seres humanos e para o meio ambiente. A ANVISA recomenda o descarte seguro e apropriado destes modelos de termômetros para uso doméstico, e este procedimento deve ser realizado através de centros de reciclagem do município do dono do termômetro (ANVISA, 2016).

b) Termômetro a álcool

Para os usuários que não pretendem utilizar os termômetros eletrônicos, os termômetros a álcool são uma alternativa segura para a substituição dos termômetros a mercúrio. O álcool é misturado com um corante, de modo que possa ser lido facilmente, e não é tóxico para o ambiente. Se o termômetro é quebrado acidentalmente, ele não oferece perigos como o de mercúrio. Ele funciona da seguinte maneira: ele mede a temperatura através de expansão térmica do etanol, e deve ser mantido no seu lugar durante vários minutos até que a expansão se estabeleça e então o resultado possa ser lido. Para reajustar o termômetro, deve-se agitá-lo até que o nível de álcool retorne à temperatura ambiente antes de usá-lo novamente.

c) Termômetros basais

Estes termômetros são altamente sensíveis e acompanham mínimas mudanças de temperatura do corpo de forma precisa. Um termômetro normal geralmente mede a temperatura com incrementos de dois décimos de um grau, enquanto esses conseguem medir com precisão de décimos de grau. Eles devem ser usados para medir a temperatura corporal central ao acordar.

Esta é a maneira mais eficiente para eliminar os fatores ambientais na temperatura do corpo, tais como o exercício ou dieta. Alterações de temperatura basal são usados para determinar a fertilidade e ovulação em mulheres que tentam engravidar. Ou seja, o rastreamento de ovulação, em vez de propósitos gerais, é feito pela medição da temperatura corporal.

d) Termômetros com infravermelho

Estes termômetros usam raios infravermelhos para medir a temperatura do corpo. A forma mais comum é usá-lo para medir a temperatura nos ouvidos, conhecido como termômetro de ouvido. Um termômetro novo e em desenvolvimento que mede a artéria temporal na testa também usa ondas de infravermelho para medir a temperatura. As principais vantagens desse tipo de termômetro são a sua precisão e a sua rapidez na obtenção do resultado, praticamente instantânea, além de não gerar desconforto que os outros modelos normalmente podem provocar na criança.

e) Termômetros Digitais

Atualmente, os termômetros digitais são amplamente utilizados, em contexto hospitalar e domiciliar, para proceder ao monitoramento de temperatura corporal. Estes termômetros, denominados termômetros de contato, possuem uma ponta com um sensor, geralmente intercambiáveis, com modelos diferentes de sensores para cada aplicação (ELKIN, 2000).

Para modelos de vidro, estes são constituídos por uma liga de vários metais no seu interior, que dilata quando há aumento de temperatura. Por ser extremamente sensível, essa expansão é medida pela variação do comprimento, numa escala graduada pode possuir uma precisão de 0,05°C. É, precisamente através da expansão do líquido, que observamos a variação da temperatura em geral (CARVALHOSA, 1994).

2.1.2. Utilização de termômetros corporais

Existem várias formas de medir a temperatura corporal de uma criança. A forma mais precisa e indicada é a retal com um termômetro digital. Mas isso pode causar certo desconforto à criança.

Existe também a medição via ouvido, mas tal procedimento não é muito recomendado para crianças, pois o acúmulo de cera e o minúsculo curvo do canal auditivo podem provocar leituras de temperatura inconsistentes.

Outra alternativa é a medição oral. Neste caso, é importante que o termômetro não se mova ao redor da boca. Além disso, abrir a boca antes do término a leitura pode causar erro de medição. Portanto, este método é uma tarefa complicada ao ser usado em crianças pequenas. Outro ponto negativo do método oral é que ele é anti-higiênico. É necessário limpar o termômetro todas as vezes antes de usá-lo.

Percebe-se que a medição nas proximidades da axila ainda é o método mais prático e menos traumático para a criança e para os pacientes acamados. Esta técnica é menos precisa que as demais, mas mesmo assim ainda fornece dados relevantes da temperatura corporal, da evolução da febre e da eficácia do tratamento. Por isso, o presente trabalho propõe o desenvolvimento de um bracelete para a medição de temperatura por contato nas proximidades da axila.

2.1.3. Medição em Tempo Real

A medição em tempo real torna possível a obtenção dos valores de temperatura corporal em qualquer lugar e em qualquer instante. Basta o bebê ou o paciente utilizar o sensor. De tempo em tempo, os dados são enviados para a internet indiretamente de tal forma que o usuário destino possa acessá-los remotamente via *smartphone* ou *tablet*.

2.2. IoT (INTERNET OF THINGS)

A Internet das Coisas é o conceito de objetos do dia-a-dia, desde máquinas complexas a dispositivos vestíveis, adquirindo dados e tomando ações através da rede, por exemplo, um escritório que utiliza sensores para automaticamente regular temperatura e intensidade luminosa, um maquinário pesado em uma linha de produção alertando que precisa de manutenção, ou, um sensor corporal informando ao seu usuário a temperatura. De forma direta, a Internet das Coisas é o futuro da tecnologia que pode tornar nossas vidas mais eficientes (VERMSAN, 2014).

Idealizada em 1991 pelo fundador da Sun Microsystems, Bill Joy, a ideia de conectar dispositivos e objetos entre si era então chamada de conexão *Device-to-Device* (D2D), onde tal conexão entre aparelhos faria parte de um conceito maior, o de “várias redes”. (ERICSSON, 2011)

Em 1999, na concepção do pesquisador britânico Kevin Ashton, criou-se o termo “Internet das Coisas”. Em 2009, ele escreveu o artigo *“That ‘Internet of Things’ Thing*”. Neste trabalho ele visualizou que, no futuro, a limitação do tempo e da rotina faria com que as pessoas se conectassem à internet por outras maneiras, e que todos os dispositivos estariam integrados à internet. Isto possibilitaria, entre outras coisas o acúmulo de dados do corpo humano, tais como temperatura, pressão arterial e batimentos cardíacos. Com tais registros, seria possível otimizar e economizar recursos energéticos e naturais. Para o especialista, tal revolução seria muito maior do que o próprio desenvolvimento do mundo conectado da forma que conhecemos atualmente (ASHTON, 2009).

As características fundamentais, (FIGURA 1) para funcionamento e devida integração dos dispositivos na era da Internet das coisas são as seguintes (HIGH, 2014):

- Interconectividade: No que diz a respeito à Internet das coisas, qualquer coisa, dispositivo ou objeto pode ser interligado com a informação global e a infraestrutura de comunicação.

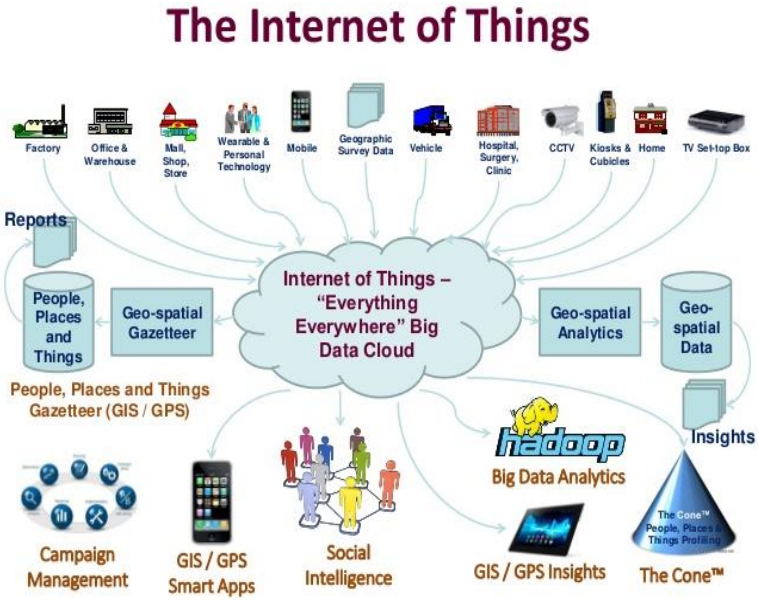
- Serviços de coisas-relacionadas: A Internet das coisas é capaz de fornecer serviços de coisas-relacionadas dentro das limitações das mesmas, tais como a proteção da privacidade e consistência semântica entre as coisas físicas e das coisas virtuais associadas. A fim de fornecer serviços de coisa relacionada dentro dos limites das coisas, tanto as tecnologias no mundo físico e no mundo da informação mudarão.

- Heterogeneidade: Os dispositivos na Internet das coisas são heterogêneos, sendo baseados em plataformas de *hardware* diversas. Eles podem interagir com outros dispositivos ou plataformas através de diferentes redes.

- Mudanças dinâmicas: O estado de dispositivos pode mudar dinamicamente, por exemplo, dormir e acordar, ligado e / ou desligado, bem como o contexto de dispositivos, incluindo a localização e velocidade. Além disso, os números de dispositivos podem alterar dinamicamente.

- Larga Escala: O número de dispositivos que precisam ser geridos e que se comunicam uns com os outros será, pelo menos na ordem de grandeza, maior do que os dispositivos ligados atualmente à internet. Ainda mais crítico será a gestão dos dados gerados e sua interpretação para fins de aplicação. Trata-se de semântica de dados, bem como a manipulação de dados de forma eficiente.

FIGURA 1 - EXEMPLOS DE APLICAÇÕES DE IoT



FONTE: FUKUSHIMA (2015).

2.3. PROTOCOLOS DE COMUNICAÇÃO

2.3.1. IPv6

A Internet começou com um projeto do Departamento de Defesa do governo norte americano que, em 1966, por meio da ARPANET, onde iniciou-se a interligação entre os computadores de pesquisa e uso militar, afim de facilitar a troca de informações entre as instalações do governo dos Estados Unidos.

No início, a ARPANET trabalhava com diversos protocolos de comunicação, com enfoque no NCP. Entretanto, em primeiro de janeiro de 1983, quando a rede atingiu a marca de 562 *hosts*, todas as máquinas da ARPANET passaram a adotar como padrão os protocolos TCP/IP. Essa mudança provocou o crescimento ordenado da rede, pois eliminou restrições dos protocolos anteriores (NIC.BR, 2012).

O protocolo IP foi definido na RFC 791 para prover duas funções básicas: a fragmentação e o endereçamento. A versão de protocolo utilizada, desde aquela época até os dias atuais, é a 4, comumente referida com o nome do protocolo de IPv4. As especificações do IPv4 reservam 32 *bits* para endereçamento, o que possibilita gerar mais de 4 bilhões de endereços distintos. Mas devido ao ritmo de crescimento da Internet e da política de distribuição de endereços, em maio de 1992, 38% das faixas de endereços classe A, 43% da classe B e 2% da classe C, já estavam sendo usados. Nesta época, a rede já possuía 1.136.000 *hosts* conectados (NIC.BR, 2012.)

Em 1993, com a criação do protocolo HTTP e a liberação por parte do governo norte americano para a utilização comercial da *Internet*, houve um salto ainda maior na taxa de crescimento da rede, que passou de 2.056.000 de *hosts* em 1993 para mais de 26.000.000 de *hosts* em 1997. O NAT, foi outra técnica paliativa desenvolvida para resolver o problema do esgotamento dos endereços IPv4 (NIC.BR, 2012).

O NAT quebra o modelo fim-a-fim da Internet, não permitindo conexões diretas entre dois hosts, o que dificulta o funcionamento de uma série de aplicações, como P2P, VoIP e VPNs.

Apesar de que a solução com NAT tenha diminuído a demanda por IPs, os problemas relacionados não foram totalmente resolvidos. Pois o uso dessa técnica reduziu apenas 14% o número de blocos de endereço pedido à IANA. Mas a necessidade de novos blocos de endereços continuou crescendo de maneira exponencial. Portanto, essas medidas tomadas foram apenas para que houvesse

mais tempo para que uma nova versão do IP fosse desenvolvida, que atende as principais características do IPv4 e além disso, fosse melhor em outros aspectos, tais como: segurança, melhoria na administração de rede, maior mobilidade, maior quantidade de endereços e capacidade de suprir as falhas apresentadas na versão anterior (NIC.BR, 2012).

Desta maneira, no final de 1993 a IETF formalizou, através da RFC 1550, as pesquisas a respeito da nova versão do protocolo IP, solicitando o envio de projetos e propostas para o novo protocolo. Esta foi umas das primeiras ações do grupo de trabalho da IETF, denominado IPng. As principais questões que deveriam ser abordadas na elaboração da próxima versão do protocolo IP foram:

- escalabilidade;
- segurança;
- configuração e administração de rede;
- suporte a QoS;
- mobilidade;
- políticas de roteamento;
- transição.
- soluções em protocolos

Diversos projetos começaram a estudar os efeitos do crescimento da Internet, sendo os principais o CNAT, o IP Encaps, o Nimrod e o Simple CLNP. Destas propostas surgiram o TUBA, que foi uma evolução do *Simple* CLNP, e o IPAE. Alguns meses depois foram apresentados os projetos PIP, o SIP e o TP/IX. Uma nova versão do SIP, que englobava algumas funcionalidades do IPAE, foi apresentada pouco antes de agregar-se ao PIP, resultando no SIPP. No mesmo período, o TP/IX mudou seu nome para CATNIP (NIC.BR, 2012).

Em janeiro de 1995, na RFC 1752, o IPng apresentou um resumo das avaliações das três principais propostas:

CANTIP - foi concebido como um protocolo de convergência, para permitir a qualquer protocolo da camada de transporte ser executado sobre qualquer protocolo de camada de rede, criando um ambiente comum entre os protocolos da *Internet*, OSI e Novell;

TUBA - sua proposta era de aumentar o espaço para endereçamento do IPv4 e torná-lo mais hierárquico, buscando evitar a necessidade de se alterar os protocolos

da camada de transporte e aplicação. Pretendia uma migração simples e em longo prazo, baseada na atualização dos *host* e servidores DNS, entretanto, sem a necessidade de encapsulamento ou tradução de pacotes, ou mapeamento de endereços;

SIPP - concebido para ser uma etapa evolutiva do IPv4, sem mudanças radicais e mantendo a interoperabilidade com a versão 4 do protocolo IP, fornecia uma plataforma para novas funcionalidades da *Internet*, aumentava o espaço para endereçamento de 32 bits para 64 bits, apresentava um nível maior de hierarquia e era composto por um mecanismo que permitia “alargar o endereço” chamado *cluster addresses*. Já possuía cabeçalhos de extensão e um campo *flow* para identificar o tipo de fluxo de cada pacote (NIC.BR, 2012).

Deste modo, a recomendação final para o novo Protocolo *Internet* baseou-se em uma versão revisada do SIPP, que passou a incorporar endereços de 128 bits, juntamente com os elementos de transição e autoconfiguração do TUBA, o endereçamento baseado no CIDR e os cabeçalhos de extensão. Após esta definição, a nova versão do Protocolo *Internet* passou a ser chamado oficialmente de IPv6 (NIC.BR, 2012).

2.3.2. BLE

A ampla difusão de dispositivos de comunicação sem fio, com preços acessíveis, trouxe para a realidade muitos serviços M2M (*Machine-to-Machine),* no qual os dispositivos trocam, entre si, informação de forma autônoma. É esperado que o número de dispositivos atinja a marca de 50 bilhões em 2020 (TOWNSEND, 2014).

Durante décadas, uma enorme variedade de tecnologias sem fio foi lançada para aplicação na área de saúde. Tais como: *ZigBee* / IEEE802.15.4, *Bluetooth, ANT, NFC* ou IrDA, todas visando baixo consumo de energia. No entanto, a tecnologia *Bluetooth Low Energy (BLE),* desenhada originalmente pela Nokia como “*Wibree*”, não foi desenvolvida como qualquer outra solução sem fio para solucionar os problemas existentes (TOWNSEND, 2014).

Esta foi completamente construída no conceito de padrão de rádio com o menor consumo possível de energia, especificamente otimizado para custo baixo, menor ocupação de banda e baixa complexidade. Estas metas de design são evidentes através da especificação central do recurso, que visa definir o BLE como um padrão

genuíno de baixo consumo de energia.

Este pode ser considerado o primeiro padrão amplamente adotado que pode realisticamente funcionar por um longo período de tempo sendo mantida apenas por uma bateria em célula do tamanho de uma moeda, feito clamado por muitas outras tecnologias sem fio (FUKUSHIMA, 2015).

O que fez do BLE uma tecnologia admirável foi a sua rápida taxa de adoção pelo mercado. Comparada com outros padrões *wireless*, o ágil crescimento do BLE é relativamente fácil de ser explicado: o BLE foi mais longe por estar intimamente amarrado ao crescimento fenomenal de *smartphones*, *tablets* e *mobile computing*.

Sendo considerada por Townsend (2014) como uma tecnologia jovem, introduzida em 2010, a mesma já possui um grande número de produtos que a possuem em seu projeto, o que a coloca muito à frente de outras tecnologias sem fio lançadas no mesmo período. O BLE oferece recursos interessantes, tais como: gerenciamento baseado na internet, detecção de localização dos dispositivos e comunicação ininterrupta mesmo no caso de comutação de rede. É importante para considerar todas as propriedades desta tecnologia e contrastá-las com as concorrentes no mercado atualmente.

Desta forma, nesse papel, tal avaliação partiu de um ponto de vista experimental, referindo-se às especificações técnicas das tecnologias. A discussão é voltada para avaliar a medida em que teses tecnológicas podem satisfazer as exigências rigorosas para cuidados com a saúde. Os padrões BLE e 6LoWPAN apresentaram maiores potenciais para essa aplicabilidade em termos de demanda de potência, baixa taxa e latência.

A Bluetooth SIG incluiu, em junho de 2010, o Bluetooth Low Energy com a versão 4.0 da Bluetooth Core Specification. O Bluetooth clássico e o BLE não são compatíveis diretamente, e os dispositivos qualificados para qualquer versão anterior ao 4.0 não podem comunicar de forma alguma com um dispositivo BLE.

O protocolo on-air, os protocolos superiores e as aplicações são diferentes e incompatíveis entre as duas tecnologias (BLUETOOTH SIG, 2016).

Existem tecnologias de soluções *Bluetooth*, diferenciadas em três modos de operação (TABELA 1).

Segundo Townsend (2014), a especificação Bluetooth 4.0 e superior define seis tipos de configuração (FIGURA 2).

Dispositivos com a tecnologia BR/EDR apenas se comunicam com dispositivos

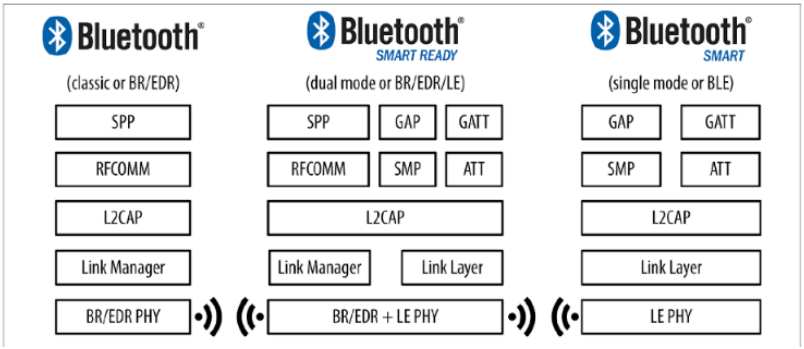
com a mesma tecnologia. Já dispositivos que funcionam em *dual-mode*, ou dispositivos *smart-ready*, se comunicam tanto com BR/EDR, como com BLE ou LE PHY.

TABELA 1 - CONFIGURAÇÕES DE ESPECIFICAÇÃO

Device	BR/EDR (classic Bluetooth) support	BLE (Bluetooth Low Energy) support
Pre-4.0 Bluetooth	Yes	No
4. Single-Mode (Bluetooth Smart)	No	Yes
4.x Dual-Mode (Bluetooth Smart Ready)	Yes	Yes

FONTE: TOWNSEND (2014).

FIGURA 2 - PADRÕES BLUETOOTH E TIPOS DE DISPOSITIVO



FONTE: TOWNSEND (2014).

2.3.3. 6LoWPAN

O conceito por trás da utilização de uma rede baseada em IP é para facilitar o modelo de conectividade entre os dispositivos. Desta forma, simplifica-se o uso de gateways complexos, e passa-se a utilizar o bem-difundido *Internet Protocol*, simplificando os bridges e roteadores, estes então bem compreendidos, desenvolvidos e disponíveis em várias tecnologias.

Adicionalmente, também existe a possibilidade de utilizar ferramentas já desenvolvidas para gerenciar, configurar e depurar estas redes, simplificando a necessidade de desenvolver várias ferramentas.

O 6LoWPAN foi desenvolvido desde o início para ser utilizado em redes

pequenas ou microscópicas de sensores. Comparando em termos de infraestrutura (TABELA 2), implementações deste protocolo possuem tamanho de 32kB de memória flash, enquanto o protocolo ZigBee podem ocupar 32kB até 90kB na memória. (MULLIGAN, 2007).

Graças também ao avanço do protocolo IPv6, foi possível eliminar a necessidade de servidores de configuração (DHCP e NAT), considerando que o endereço IPv6 possui 128 *bits* de endereço, ao contrário do protocolo anterior, IPv4. Desta forma, o protocolo da sexta versão é capaz de fornecer 677.10^{21} endereços por metro quadrado no planeta Terra.

Ambos, BLE e 6LoWPAN, são competidores fortes na área de protocolos de comunicação sem-fio com baixo consumo de energia. Apesar do BLE oferecer mais características em alguns aspectos, e este protocolo também ter a tendência de expandir rapidamente no mercado. Mas comparado com o protocolo baseado no IPV6, o 6LoWPAN foi amplamente testado em aplicações na área de saúde, sendo considerado por Tabish (2013) o protocolo mais adequado para sensores corporais.

TABELA 2 - COMPARAÇÃO ENTRE OS PROTOCOLOS

	Zensys	**6LoWPAN**
Code size with mesh	32kB	22kB code
Size w/o mesh	Not Possible	12kB
RAM Requirements	<2kB	4kB
Header Overhead	Proprietary	2 to 11 bytes
Network Size	232	N/A
RF Radio Support	Proprietary	802.15.4 ++
Transport Layer	Proprietary	UDP/TCP
Mesh Network Support	Zensys	Many
Internet Conectivity	Zensys Gateway	Bridge/Router

FONTE: Mulligan, 2007.

2.3.4. WI-FI

Wi-Fi é um termo utilizado por produtos certificados que pertencem à classe de dispositivos de rede local sem fios (WLAN), baseados no protocolo padrão IEEE

802.11. Ele opera em faixas de frequências que não necessitam de licença para instalação e/ou operação. As frequências usadas são livres de licença, o usuário não paga nenhuma taxa. Estes fatos o torna atrativo. No entanto, para uso comercial no Brasil, é necessário o equipamento tenha sido analisado, avaliado e obtido um certificado de homologação pela ANP - Agência Nacional de Telecomunicações.

Para se ter acesso à internet através de rede Wi-Fi, o dispositivo deve estar no raio de ação ou área de abrangência de um ponto de acesso (AP) ou local público onde opere rede sem fios livre.

O ponto de acesso transmite o sinal sem fios numa pequena distância, geralmente de até 100 metros, mas se a rede for do padrão IEEE 802.11n a distância pode chegar até 300 metros.

2.4. DISPOSITIVOS SoC (SYSTEM-ON-CHIP)

System On Chip (SOC) ou, em português, sistema-em-um-chip, se refere a todos os componentes de um computador, ou qualquer outro sistema eletrônico, em um único circuito integrado (CI).

Ele pode conter funções digitais, analógicas, mistas e, muitas vezes, de radiofrequência, tudo em um CI. Uma típica aplicação é na área de sistemas embarcados (DUARTE, 2014).

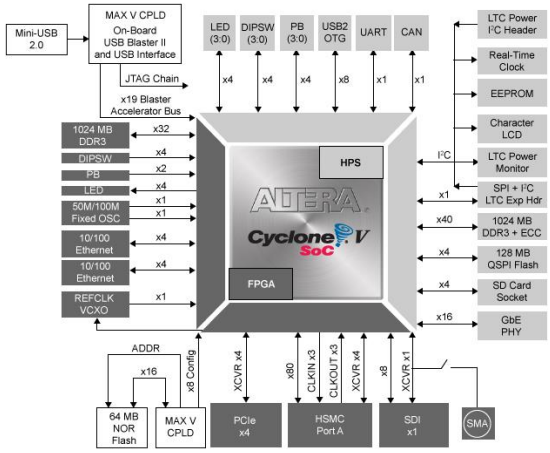
Um SoC típico consiste em um microcontrolador, um microprocessador ou um núcleo DSP com os seus periféricos

Alguns SoCs são chamados de sistema multiprocessador no *chip* (MPSOC). Possuem: mais do que um processador de núcleo; blocos de memória incluindo um sistema de ROM, RAM EEPROM e Memória *flash*; osciladores e anéis de bloqueio de fase; periféricos incluindo temporizadores, temporizadores em tempo real e geradores de redefinição de ativação; interfaces externas, incluindo os padrões da indústria, tais como USB, *FireWire*, Ethernet, USART, SPI; interfaces analógicas incluindo ADCs e DACs; reguladores de tensão e circuitos de gerenciamento de energia (DUARTE, 2014).

O uso de processadores embarcados em dispositivos lógicos programáveis vem crescendo muito nos últimos anos. Para atender esta demanda crescente, a Altera lançou a linha SoC de FPGAs (*Field Programmable Gate Arrays*), que são FPGAs com processador e um conjunto de periféricos já implementados em hardware,

que permitem atingir alto desempenho nos projetos, tanto do software rodando no processador quanto de sistemas implementados em hardware no FPGA (FIGURA 3).

FIGURA 3 - ESQUEMÁTICO DE UM KIT DE DESENVOLVIMENTO DE SOC



FONTE: ALTERA (2016).

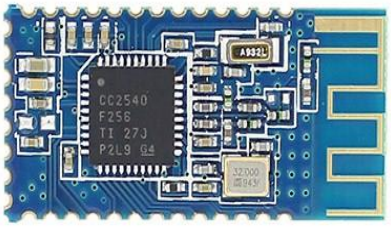
Ao longo da última década, a porcentagem de projetos em FPGA que usam algum tipo de processador embarcado subiu de menos de 10% dos projetos em 2003, para mais de 50% dos projetos em 2013 e acredita-se que esse aumento seja uma tendência. A demanda por maior capacidade de processamento de *software*, maior número de núcleos nos processadores do sistema e redução da área de placa e do consumo do sistema, para viabilizar aplicações embarcadas, levou a Altera a desenvolver a linha SoC de FPGAs (DUARTE, 2014**).**

A Texas Instruments também tem desenvolvido seus modelos de SoC. O modelos que foram usados como parte do presente trabalho são o CC2541 da Texas Instruments (FIGURA 5), presente no módulo BLE HM -10 (FIGURA 4), e o CC2650 também da Texas, presente no LaunchPad, utilizado na segunda parte do trabalho. O primeiro é uma solução SoC otimizada que possui BLE e opera em até 2,4 GHz, além de controlador de rede de baixo custo. O CC2541 é uma combinação de um excelente desempenho de transmissor e receptor de radiofrequência e um núcleo de microcontrolador 8051 melhorado. Possui memória *flash* programável, 8 kB de RAM e muitos outros periféricos. O CC2541 é altamente indicado para sistemas onde o consumo ultrabaixo de energia é pré-requisito (TEXAS INSTRUMENTS, 2016).

O segundo, é um *kit* de desenvolvimento que suporta vários protocolos, tais como BLE, 6LoWPAN e ZigBee. O circuito utilizado é gerenciado por um processador

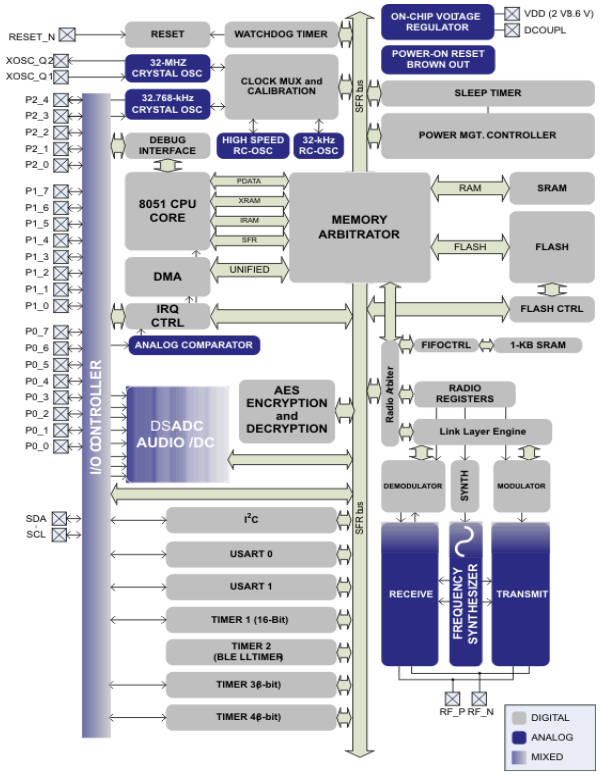
32-*bits* ARM Cortex M3, com velocidade de 48MHz com uma série de periféricos também com baixo consumo. A opção por utilizar este componente está na sua capacidade de comunicação com sensores para coletar de dados analógicos e os converter em digital, via função AD, enquanto o resto do sistema está em modo de economia de energia, além da compatibilidade com IP versão 6.

FIGURA 4 - BLE COM PROCESSADOR CC2541



FONTE: GitHub (2016).

FIGURA 5 - DIAGRAMA DO CC2541



FONTE: Texas Instruments (2016).

2.4.1. ESP8266

O microcontrolador ESP8266 integra um processador RISC 32-bits Tensilica

L106, com extra baixo consumo de energia e velocidade de *clock* de 160Mhz. O RTOS e o *stack* Wi-Fi possibilitam que 80% do poder de processamento esteja disponível para programação e desenvolvimento de aplicações do usuário. Por ser desenvolvido para dispositivos móveis, tecnologias vestíveis e aplicações IoT, este microcontrolador atua majoritariamente em 3 modos de operação: ativo, espera e espera profunda. Isto possibilita aplicações projetadas para uso com bateria sejam executadas por mais tempo.

2.5. SERVIDOR APACHE

Trata-se de um servidor muito popular e é usado principalmente na plataforma Linux. Assim como qualquer servidor, sua principal função é disponibilizar a página e o conteúdo na internet para internauta possa acessá-lo em qualquer lugar do mundo. Graças a ele, o usuário pode enviar mensagem, e-mail, ler notícia, fazer compra online, entre as outras funções. Apesar de que o Apache seja livre, ele é distribuído como sublicença do GNU. No entanto, qualquer pessoa tem o acesso ao seu código fonte e pode modificá-lo de acordo com a necessidade. Portanto, um servidor gratuito e funcional.

Por esse motivo, em dezembro de 2007, uma pesquisa foi realizada, onde consta cerca de 47% dos servidores no mundo são representados pelo Apache. Em 2012, essa porcentagem aumentou para quase 55% (APACHE, 2017).

Esse aumento de popularidade do Apache pode ser explicado também através da sua compatibilidade com o HTTP versão 1.13. Sendo que suas funcionalidades e compatibilidades são mantidas através de uma estrutura de módulos. Isso permite que o usuário ou internauta escreve o seu próprio módulo por meio da API do software. Ele está disponível para plataformas Windows, Novell *Netware*, OS/2, Unix e Linux, onde é largamente utilizado. Além disso, o Apache possui um módulo chamado de mod_ssl, onde aumenta a capacidade do servidor a atender solicitações usando protocolo HTTPS. Sendo que este protocolo usa sua camada SSL para realizar criptografia dos dados em tráfico. Portanto, aumenta a segurança na troca de informações entre o cliente e o servidor.

Em relação ao hardware, o suporte ao servidor depende muito do uso ou da aplicação, mas em geral, um simples PC com processador Pentium e 64 MB de memória RAM já basta. Lembrando que hoje em dia, qualquer computador pessoa é

capaz de colocar um servidor Apache em funcionamento. Até mesmo um *Raspberry Pi*, o que será utilizado para criar e colocar na internet o servidor Apache, onde qualquer usuário cadastrado pode acessar, em qualquer lugar do mundo, o histórico das medições de temperatura corporal, obtidas via sensor de temperatura.

2.6. RASPBERRY PI

Raspberry Pi (FIGURA 6) é um computador de tamanho de um cartão de crédito, que pode ser conectado ao TV, teclado, *mouse* e demais comuns equipamentos periféricos. Esse projeto de Raspberry Pi começou a ser desenvolvido pela fundação de mesmo nome em fevereiro de 2012 e com objetivo de baratear o custo relacionado ao computador cujo foco é na área de educação.

FIGURA 6 - RASPBERRY PI 3 MODELO B

FONTE: Os Autores (2017).

Com este dispositivo, o usuário é capaz de fazer edição de planilha, assistir vídeos, ouvir música, navegar na internet, editar textos e demais funções básicas que qualquer PC possa realizar. Por ter um *hardware* muito limitado, algumas tarefas, que exigem muito poder de processamento, como em jogos de alta qualidade de imagem e na edição de vídeo, está fora de alcance dele. Até porque o objetivo principal é volta para educação, com pretensão de levar tecnologia e conhecimento às escolas menos favorecidas economicamente.

Apesar da simplicidade, o *hardware* dele suporta diversas distribuições Linux. Sendo que o sistema operacional é instalado no cartão de memória, uma vez que o

dispositivo não possui disco rígido próprio.

Além disso, todos os dados relacionados ao dispositivo podem ser encontrados no site do próprio fabricante, onde consta o código aberto para ser consultado. Isso permite que o usuário possa fazer as devidas modificações, outras versões de tal maneira que contribui ainda mais para baratear o dispositivo original.

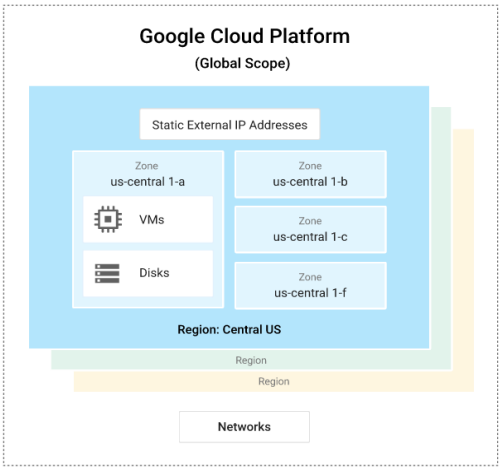
O Raspberry Pi 1 é baseado em SoC Broadcom BCM2835, que o Raspberry Pi 3 model B contém um processador 1.2GHz, 64-bit e quatro núcleos ARMv8, 1 GB de memória de RAM, Bluetooth 4.1. O projeto não inclui um disco rígido, mas possui uma entrada de cartão SD para armazenamento de dados (HAGUE, 2008).

2.7. AMBIENTES *CLOUD*

2.7.1. PLATAFORMA CLOUD GOOGLE

O Cloud Platform consiste em um conjunto de ativos físicos, como computadores e unidades de disco rígido, além de recursos virtuais, como máquinas virtuais, contidos em data centers do Google em todo o mundo. Cada local de data center está em uma *região* global. Entre as regiões, estão: a central dos EUA, a Europa Ocidental o Leste da Ásia. Cada região é uma coleção de *zonas*, isoladas entre si dentro da região. Cada zona é identificada por um nome que combina um identificador de letra com o nome da região (FIGURA 7).

FIGURA 7 - ESCOPO GLOBAL DO GCP



FONTE: Google Cloud Platform (2017).

Essa distribuição de recursos oferece diversas vantagens, inclusive uma redundância em caso de falha e latência reduzida localizando recursos mais próximos dos clientes. Essa distribuição também introduz regras sobre como recursos podem ser usados juntos.

O escopo de uma operação varia de acordo com que tipo de recursos você está trabalhando. Por exemplo, criar uma rede é uma operação global porque uma rede é um recurso global e reservar um endereço IP é uma operação regional porque o endereço é um recurso regional.

O Google Cloud Platform oferece uma interface gráfica do usuário que torna fácil e simplificado a utilização para gerenciar os projetos e os recursos do Cloud Platform. Ao usar o Cloud Platform Console, existe a possibilidade de criar um novo projeto ou escolher um existente, utilizando os recursos criados no contexto desse projeto.

Alguns recursos podem ser acessados por qualquer outro recurso, entre regiões e zonas. Entre esses *recursos globais* estão imagens de disco pré-configuradas, instantâneos de disco e redes. Alguns recursos só podem ser acessados por recursos localizados na mesma região. Entre esses *recursos regionais* estão endereços IP externos estáticos. Outros recursos só podem ser acessados por recursos localizados na mesma zona. Entre esses *recursos por zona* estão instâncias de VM, tipos e discos.

2.7.2. THINGSPEAK

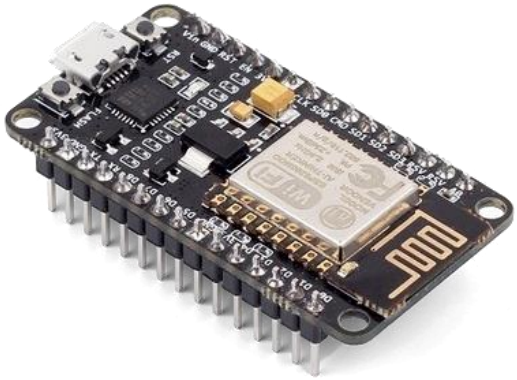
ThingSpeak é um serviço que oferece uma infraestrutura de web e um protocolo de comunicação baseado em HTTP para envio e recebimento de dados gerados em Arduino ou qualquer outro dispositivo com recursos para comunicação em rede. Pode-se dizer que qualquer dispositivo com conectividade para a Internet com serviço http e capaz de realizar GET e POST pode se comunicar com o ThingSpeak. Com esse serviço é possível por exemplo usar o Arduino para gerar um tweet, criar uma rede social de dispositivos ou controlar a distância motores e outros dispositivos.

2.8. NODEMCU

O nodeMCU (FIGURA 8), um kit de desenvolvimento de código aberto, foi

concebido visando auxiliar no desenvolvimento fácil de protótipos com aplicações IoT. Possuindo como sua unidade de controle principal o ESP8266, o kit disponibiliza as portas de entrada e saída do ESP8266 e simultaneamente adiciona a interface de comunicação USB a unidade principal.

FIGURA 8 - KIT NODEMCU



Fonte: RoboCore (2016).

2.9. PLATFORMIO

PlatformIO é uma IDE flexível concebido para o mercado de desenvolvimento de sistemas embarcados. Assim, este é capaz de compilar códigos multi-plataforma, independente do sistema operacional utilizado. Assim, além de tornar o desenvolvimento mais ágil para aqueles que utilizam múltiplos sistemas operacionais. A IDE suporta 423 placas embarcadas, 22 plataformas de desenvolvimento e 12 *frameworks.*

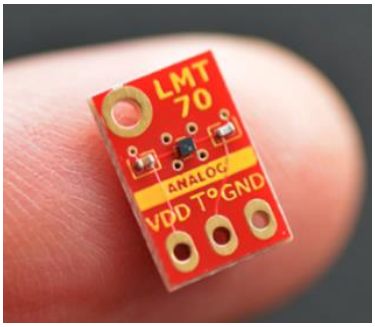
2.10. SENSOR DE TEMPERATURA LMT70

LMT70 (FIGURA 9) é um sensor analógico preciso de temperatura. Além de ser um dispositivo de baixo consumo, ele também apresenta o tamanho de apenas 0,88 mm x 0,88 mm, o que é um ótimo sensor para aplicação em IOT e equipamento na área médica.

De acordo com o fabricante, na faixa de temperatura de 20°C a 42°C, o sensor apresenta um erro de apenas ±0,05°C (FIGURA 10), o que é excelente para este projeto, em que o objetivo é medir a possível febre do corpo humano, cujo temperatura

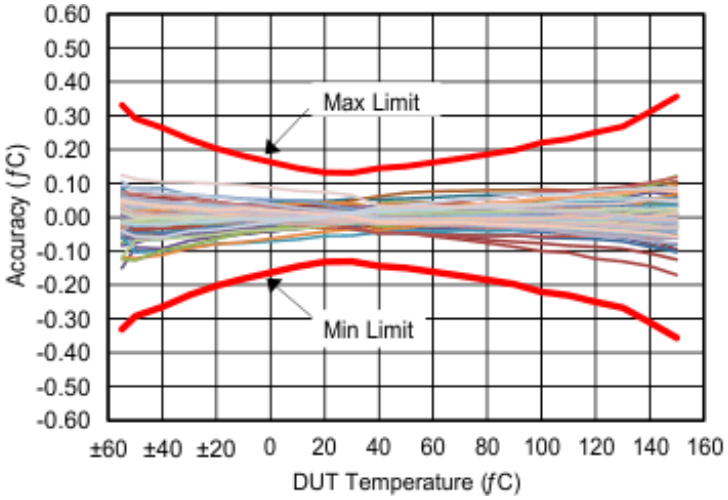
se encontra na faixa anteriormente citada.

FIGURA 9 - SENSOR DE TEMPERATURA LMT70



FONTE: Tindie (2017).

FIGURA 10 - PRECISÃO DO LMT70 EM FUNÇÃO DA TEMPERATURA



FONTE: Texas Instruments (2017).

Além disso, a sua saída fornece uma tensão analógica, isto é, o sensor converte automaticamente a temperatura, onde se encontra, em tensão (mV). Depois da conversão, a TI recomenda o uso da função de transferência da saída (fornecida pelo fabricante) para fazer a transformação da tensão para temperatura em grau Celsius correspondente.

As principais características do sensor LMT70 são:

- Precisão: ±0,05°C (valor típico) do 20°C ao 42°C
- Faixa da temperatura de operação: do −55°C ao 150°C
- Ganho do sensor (tensão/temperatura): -5,19 mV/°C
- Impedância de saída: 80 Ω

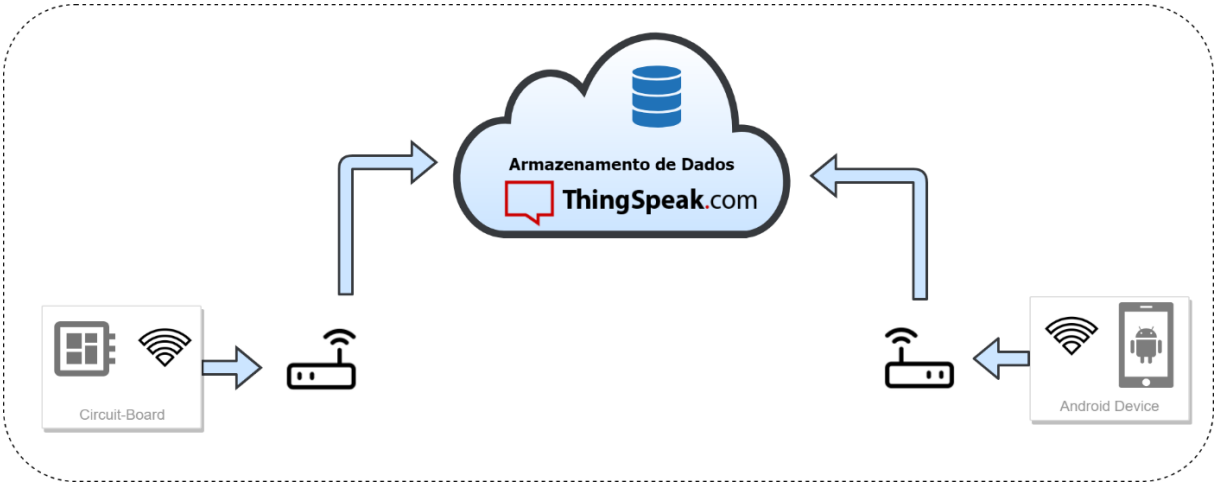
- Faixa de tensão de alimentação: 2,0 V a 5,5 V
- Corrente de baixo consumo: 9,2 µA (valor típico) 12 µA (valor máximo)

3. DESENVOLVIMENTO

3.1. DIAGRAMA FINAL DO BRACELETE

O termômetro digital em forma de bracelete é alimentado por duas baterias de íon de Lítio. Ele será controlado por um SoC ESP-WROOM-02, da EspressIf, o qual possui módulo Wi-Fi embarcado, e fornece os dados de temperatura para a plataforma ThingSpeak, da MathWorks, e é acessível por aplicativo em sistemas Android, *smartphones* ou *tablets*. A plataforma na nuvem armazena o histórico e é responsável por fornecer para a aplicação móvel. A temperatura é obtida por um circuito integrado transdutor de temperatura LMT70, e utiliza a comunicação Wi-Fi para enviar dados à nuvem. O diagrama (FIGURA 11) é a base para desenvolvimento de um protótipo.

FIGURA 11 - DIAGRAMA FINAL DO BRACELETE



FONTE: Os Autores (2018).

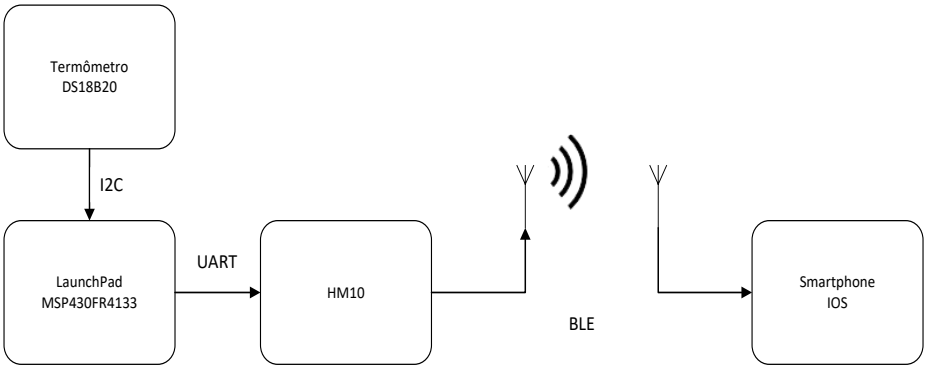
3.2. PROJETOS PRELIMINARES

O projeto foi concebido desde o início com a proposta de criar um produto inovador e com potencial de mercado, para resolver os problemas do cotidiano da população em geral. Para alcançar a proposta atual da arquitetura, meio de comunicação e solução IoT, foram utilizados vários microcontroladores e sensores, de forma a avaliar qual seria a melhor solução, mais robusta e de fácil desenvolvimento para esta aplicação.

Primeiramente, foi desenvolvido um protótipo para avaliação da viabilidade do projeto e testes preliminares da aquisição da temperatura e da comunicação via BLE, com um módulo do Arduino HM-10, sensor de temperatura DS18B20 e o microcontrolador MSP430.

O diagrama em blocos do sistema (FIGURA 12) mostra como foi elaborado o circuito.

FIGURA 12 - DIAGRAMA EM BLOCOS IDEALIZADO INICIALMENTE



FONTE: Os Autores (2016).

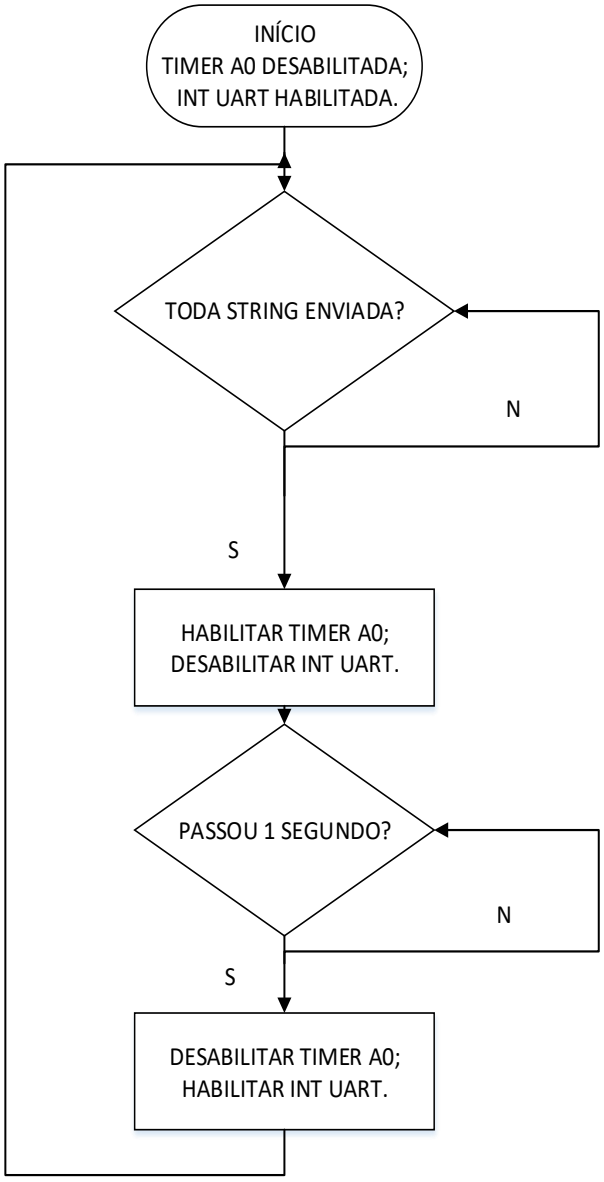
Este sensor de temperatura, foi escolhido por possuir exatidão garantida de 0,5ºC para toda a sua faixa de medição, alimentação de 3 a 5,5 Volts e faixa de medição de -55 a +125°C, já retornando o valor de temperatura convertido digitalmente em graus Celsius (MAXIM INTEGRATED, 2016),

Para testes iniciais, o código do MSP430 foi adaptado para tornar possível a leitura dos dados do sensor e a transmissão dos valores corretos para o módulo BLE. Sendo que uma das características é tornar o projeto testável de maneira sem fio. Caso venha ocorrer algo errado ou um obstáculo que inviabiliza o projeto, seria muito mais fácil de trocar a plataforma ou fazer devidas modificações. O diagrama de conexões (FIGURA 18) ilustra como esses passos foram efetuados.

Na etapa de utilização do sensor DS18B20, foram utilizadas inicialmente as funções escritas por Kurup (2016). No entanto, durante os testes, percebeu-se que a leitura de temperatura no sensor falhava em alguns momentos (FIGURA 14). Segundo o *datasheet* da fabricante (MAXIM INTEGRATED, 2016), para cada instrução enviada ao DS18B20, é necessário um intervalo específico de tempo, para que seja processada a informação pelo sensor. Considerando tal informação, os intervalos de atraso entre cada instrução foram reajustados, resolvendo a ocasional falha de leitura.

A calibração de um sensor de temperatura deve ser realizada pelo método de comparação com um instrumento de referência, rastreados a padrões nacionais ou órgãos de reconhecida credibilidade (ABNT, 2000).

FIGURA 13 - FLUXOGRAMA DAS INTERRUPÇÕES



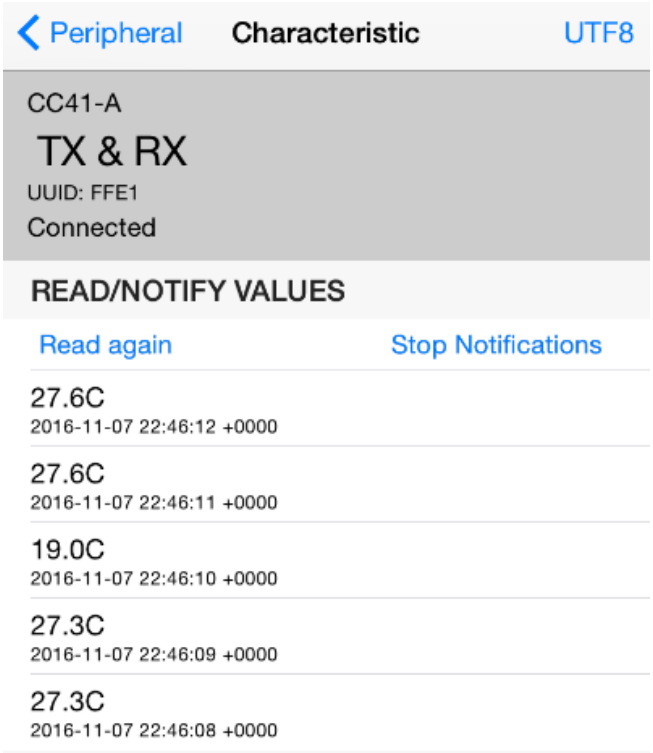
FONTE: OS AUTORES

O foco principal de projeto preliminar projeto visava a comunicação entre dispositivos com baixo consumo de energia. Nesta fase de projeto, a precisão não é o objetivo principal. Considerando isto, foi utilizado um referencial não-ideal, um termômetro doméstico, da Geratherm Medical, modelo GT2038 (FIGURA 16), com precisão de 0,1ºC no intervalo de 32ºC à 43,9ºC, para medir o erro de temperatura do

DS18B20.

Foram, então, medidos os valores de temperatura dos dois sensores, 15 vezes, (FIGURA 38).

FIGURA 14 - ERRO DE LEITURA

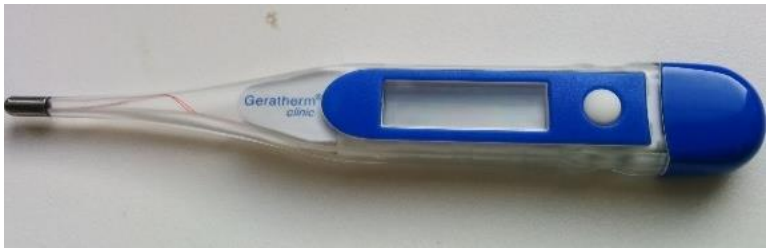


FONTE: Os Autores (2016).

FIGURA 15 - SENSOR DS18B20

FONTE: Os Autores (2016).

FIGURA 16 - TERMÔMETRO GERATHERM GT2038



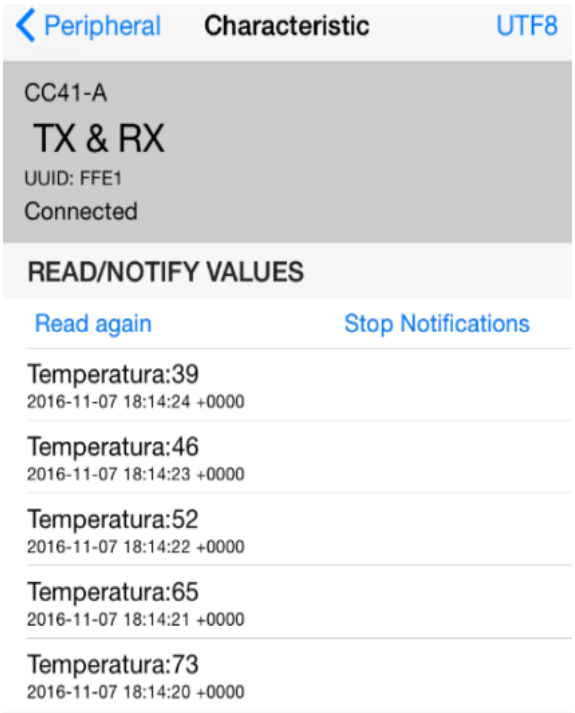
FONTE: Os autores (2016)

Desta forma, foi elaborado um novo escopo, procurando propiciar o acesso de vários dispositivos ao termômetro, mas ainda assim mantendo sua robustez e baixo consumo de energia, utilizando um protocolo mais difundido atualmente, o IP. Assim, foi efetuada a seleção do protocolo de comunicação 6LoWPAN.

Por segundo, foi substituído o MSP430 para o microcontrolador CC2650, para utilização do protocolo 6LoWPAN. Creditava-se que este *chipset* poderia eliminar por completo o módulo HM10, e realizasse por si só a comunicação com a Internet via protocolo 6LoWPAN.

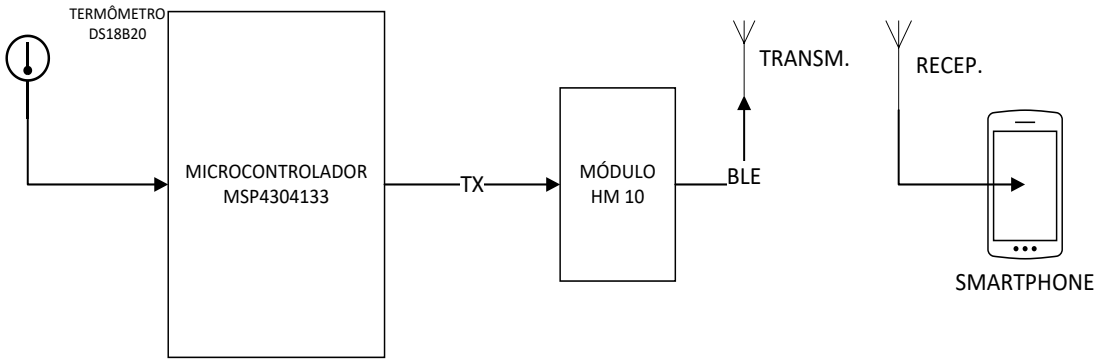
O diagrama em blocos do sistema (FIGURA 19) demonstra como foram estipuladas as conexões para desenvolvimento com este microcontrolador.

FIGURA 17 - LEITURA DO POTENCIÔMETRO NO *SMARTPHONE*



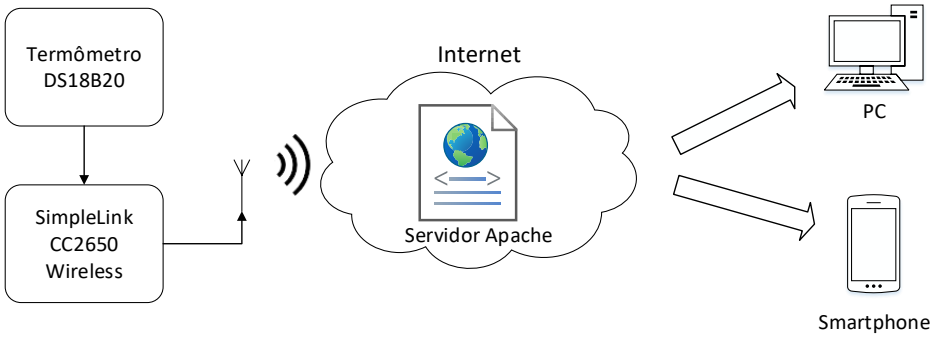
FONTE: Os autores (2016).

FIGURA 18 - DIAGRAMA DE CONEXÕES MSP430 COM DS18B20



FONTE: Os autores (2016).

FIGURA 19 - DIAGRAMA DE CONEXÔES CC2650 COM SENSOR DS18B20



FONTE: Os Autores (2017).

Nesta nova base do projeto, houve a necessidade de conhecer a funcionalidade básica do *kit* CC2650 LaunchPad e familiarizar com a nova linguagem de programação – C# (C Sharp, uma linguagem orientada a objeto), assim como estudar seu *hardware*: conversor analógico digital, temporizador, pinos de acesso, UART etc. Para isso, foi necessário o estudo dedicado no documento *User Guide* do microcontrolador para melhor compreendê-lo.

Para funcionamento adequado do microcontrolador, foi necessário o desenvolvimento de um servidor *web* local, hospedado em um Raspberry Pi, para recebimento dos dados enviados pelo LaunchPad CC2650 via protocolo 6LoWPAN.

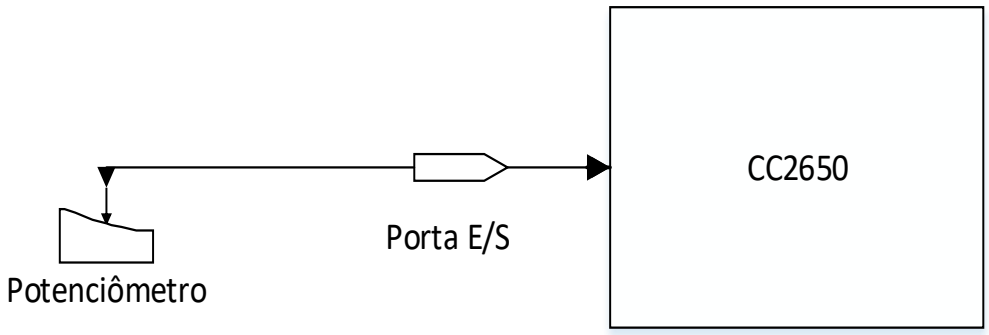
Foram alterados os parâmetros de endereço IP e portas no arquivo de inicialização do mesmo, de forma que este respondesse apenas requisições TCP para IPv4 e IPv6.

Configurado o servidor Apache no Raspberry Pi, iniciou-se o desenvolvimento da interface do usuário com o sistema responsável por gerenciar as informações

enviadas do bracelete.

Paralelamente, foi implementado no código do CC2650 o uso do *Timer* para gerar uma interrupção, em que a função de leitura do sensor de temperatura, e a leitura analógica-digital do CC2650 para efetuar a leitura do potenciômetro que simula um sensor de temperatura (FIGURA 20), e o resultado da conversão é enviado para console do CCS.

FIGURA 20 - LEITURA ANALÓGICA-DIGITAL COM CC2650



FONTE: Os Autores (2017).

3.3. PROJETO ATUAL

Para o microcontrolador CC2650, devido ao seu sistema RTOS da Texas e a necessidade de adição de plataforma extra para comunicação sem fio, o que torna sistema complexa e inviável para implementação do protótipo no futuro próximo. Foi necessário, portanto, a mudança do Microcontrolador. Na seleção por um novo microcontrolador, foram estabelecidos alguns requisitos, como:

- Aberto a comunidade para desenvolvimento;
- Menor complexidade no sistema;
- Consumo menor de energia;
- Preço.

Assim, foram reavaliados os microcontroladores utilizados para este projeto: MSP430, CC2650 e ESP8266 (TABELA 3).

Este último provou-se estar amplamente difundido no mercado, e possuir inúmeras variantes: Wemos, Sparkfun, Adafruit.

Por esse motivo o projeto foi modificado com ESP8266 para presentes necessidades (FIGURA 11).

TABELA 3 – COMPARAÇÃO DE MICROCONTROLADORES

			
	MSP430FR4133	**CC2650**	**ESP-WROOM-02**
Processador	16-Bit RISC	ARM® Cortex®-M3	Tensilica L106 32-bit RISC
Frequência	Até 16 MHz	Até 48 MHz	Até 160 MHz
Tensão de Operação	1,8 ~ 3,6V	1,8 ~ 3,8V	2,7V ~ 3,6V
Corrente em Standby	1 µA	1 µA	20 µA @ 2.5V
Comunicação	Via módulo externo	Bluetooth, ZigBee®, 6LoWPAN ($)	Wireless 802.11n
Preço	$2,57	$4,00	$2,26
Tamanho	8,6mm*13mm	7,3mm*7,3mm	18mm*20mm
Módulo externo	Sim	Sim	Não

FONTE: OS AUTORES (2018).

3.3.1. Sensor LMT70

Para realizar a comparação entre diversos tipos de sensores de temperatura, foi necessário os inserir em um ambiente controlado termicamente. Nesse caso, foi utilizada uma sala quente onde o calor é gerado por uma fonte térmica, e que a temperatura é controlada em torno de 40°C. Pois a sala possui um sistema que monitora a temperatura dela. Uma vez que a temperatura passe dos 40°C, a fonte térmica é desligada e o sistema de ventilação é acionado até que a sala esfria até 38°C, quando o mesmo é então desligado e a fonte térmica é acionada novamente.

Esta sala quente encontra-se na Grupo Lumicenter Lighting – uma empresa no ramo de fabricação e de venda de luminária a LED e possui uma equipe especializada para desenvolvimento de drivers (fonte de corrente para alimentação das luminárias). Sendo que a principal função da sala quente é testar a operacionalidade das luminárias e dos drivers sob condição estresse térmica.

3.3.1.1. Teste de Encapsulamento

Para a fabricação final de um protótipo funcional, é necessário o

encapsulamento do sensor de temperatura, para isolamento e durabilidade do mesmo. Assim, antes de encapsular este sensor, foram realizados testes utilizando dois sensores LM35, um encapsulado e outro não, para avaliação da eficiência do processo.

Assim, utilizando uma resina produzida pelo Grupo Lumicenter Lightning, fabricada localmente pela empresa para encapsular e isolar componentes com risco de sobreaquecimento em seus produtos, esta resina provou-se eficiente e eficaz em seu propósito, demonstrando ínfima variação de temperatura entre um dispositivo e outro.

3.3.1.2. Criação da Pulseira

Para fabricação da pulseira foram aproveitados materiais já presentes no mercado, destinados a crianças e bebês, de forma a não gerar incômodo ou desconforto durante a utilização. Sendo assim, utilizando uma pulseira segurança infantil (FIGURA 21), fabricada em tecido e com ajuste em velcro, para colocação do sensor.

FIGURA 21 – PULSEIRA DE SEGURANÇA INFANTIL

FONTE: ELO7 (2018).

Nesta etapa, devido ao tamanho do dispositivo, optou-se por colocar o sensor encapsulado distante do microcontrolador, de forma que fios condutores foram

inseridos ao longo do tecido da pulseira, deixando o transdutor de temperatura em contato com a parte interior do antebraço do paciente em questão.

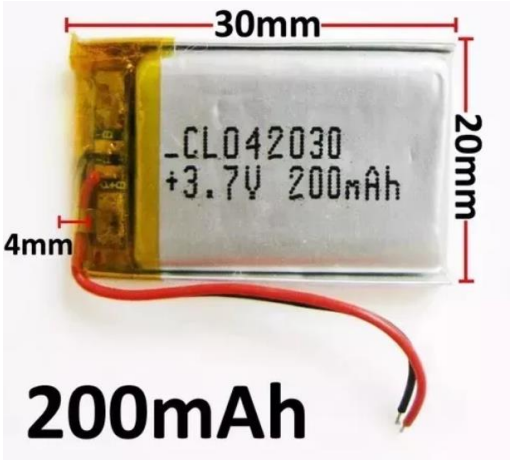
3.3.2. Bateria LIPO

O mundo dos eletrônicos portáteis é movido por baterias, por melhor que seja a sua tecnologia. Em termos de tipos de bateria, há incontáveis modelos, tamanhoe tipo. Um deles é do tipo íon-polímero (Li-Po).

Um dos componentes da bateria, o lítio, é o atual elemento-base dos principais modelos de baterias presentes no mercado. Por uma questão simples: ele é o mais leve entre todos os metais conhecidos, além de ter o melhor potencial eletroquímico e a melhor relação entre peso e capacidade energética. A bateria, de qualquer tipo, é formada por eletrodos positivos (os cátodos) e negativos (os ânodos). Nas baterias de lítio, esse elemento serve como ânodo, deixando o papel de cátodo para os elementos metálicos nas baterias de Li-Ion e polímeros (que podem ser a base de cobalto, fosfato ou manganês) nas de Li-Po.

Outra característica das Li-Po é que possuem uma densidade muito alta, isso quer dizer que a bateria tipo Li-Po, com mesma quantidade de massa, mesmo sendo mais fina, tem uma capacidade maior de fornecer energia em relação à bateria de outra tecnologia. Por esse motivo, a bateria escolhida para presente projeto é tipo Li-Po de 3,7V (), além de ter o tamanho ideal para dispositivo portátil com presente termômetro.

FIGURA 22 – BATERIA LIPO 200MAH



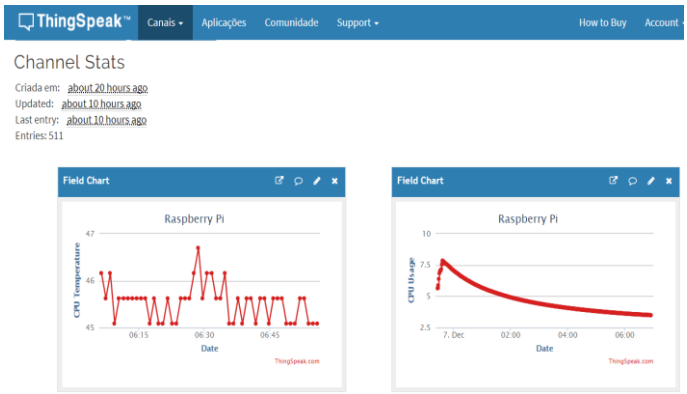
FONTE: MERCADOLIVRE (2018).

3.3.3. ThingSpeak

Utilizando a plataforma *open-source* ThingSpeak, foi possível simplificar toda a parte do *front-end* e *back-end* do projeto, realizando apenas a configuração da conta e dos canais na plataforma, nos quais os dados serão recebidos e armazenados. Assim, o próprio ThingSpeak elabora gráficos e códigos para análise destes valores no programa MATLAB (FIGURA 23).

Assim, foi desenvolvido um *script* em Python (APÊNDICE 9) que realiza a estruturação *web* e envia o valor da temperatura da CPU e sua utilização em porcentagem para o canal do ThingSpeak, que armazena e demonstra os dados na página do usuário.

FIGURA 23 - STATUS DO CANAL NO THINGSPEAK



FONTE: ThingSpeak (2017).

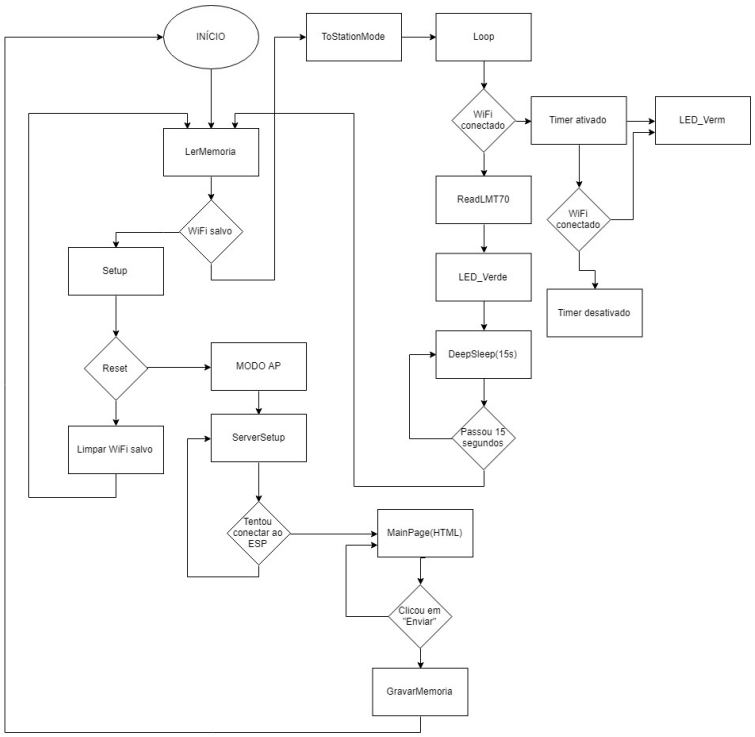
3.3.4. ESP8266

Nesta fase, utilizando a IDE do PlatformIO, modularmente. Ou seja, primeiramente, desenvolve-se a parte de comunicação e autenticação na rede Wi-Fi, utilizando funções definidas na biblioteca da própria IDE. Por segundo, é incluída a parte de autenticação e comunicação do ThingSpeak, de forma a enviar dados de leitura de um pino analógico do nodeMCU para a plataforma da MathWorks.

3.3.4.1. Desenvolvimento ESP8266

A (FIGURA 24) apresenta o fluxograma do firmware desenvolvido para o ESP8266:

FIGURA 24 – FLUXO DE CÓDIGO DO FIRMWARE PARA TERMÔMETRO



FONTE: OS AUTORES (2018).

Ao iniciar o código, o sistema lê a região de memória para verificar a existência de nome ou senha de WiFi previamente salvo. Procura pelo o nome e senha de um sinal que já foi conectado anteriormente. Se o resultado da busca for positivo, o ESP entra no modo cliente, ou seja – solicita conexão a esta rede Wi-Fi. Se não houver nenhuma rede cadastrada, o ESP entra no modo AP (ponto de acesso), comportando-se como um servidor e criando uma rede privada.

Isso permite que o usuário, usando um *smartphone* com o aplicativo, conecte-se a ele e realize a configuração de rede.

A função LerMemoria possui como objetivo buscar dados como o nome e a senha da rede previamente gravadas. Em caso positivo, ele retorna esses parâmetros; caso negativo, ele devolve ao sistema dois vetores vazios.

Na função *Setup*, os pinos 5 e 14 do ESP são declarados como saídas para acionar LEDs, e, ao mesmo tempo, colocar o microcontrolador em modo AP. Nesta situação, a subfunção ServeSetup é chamada para verificar se há cliente está

tentando se conectar a página web local ou não.

Uma vez acessada a página web local (192.168.4.1), o conteúdo da página é enviado e é aguardado que o cliente forneça o nome e a senha do WiFi ao qual o usuário deseja conectar o dispositivo.

Concluído o fornecimento das informações, o usuário deve clicar no botão "*Submit*". Em seguida, o microcontrolador copia os dados fornecidos e os grava na memória do mesmo.

Após a gravação, o sistema volta ao início do código e a função LerMemoria é chamada novamente. Mas desta vez, por já possuir os parâmetros da rede, o ESP entra no modo *Station* – ele se comporta como cliente, e tenta conectar ao sinal emitido pelo modem.

Dentro da função principal Loop, inicialmente é verificado se o dispositivo está conectado à rede Wi-Fi ou não. Em caso positivo, a função ReadLMT70 é chamada para ler o valor do termômetro, isto é: a tensão fornecida pelo sensor LMT70. Quando a leitura é realizada, a tensão lida é convertida para temperatura em grau Celsius via função de transferência (FIGURA 25), fornecida pelo fabricante do sensor.

FIGURA 25 – FUNÇÕES DE TRANSFERÊNCIA (LMT70)

```
if (adc_value >943 && adc_value <=995)
{
  temperatura = -0.193 * adc_value + 212.009;
}
else if (adc_value >891 && adc_value <=943)
{
  temperatura = -0.19212665 * adc_value + 211.2190437;
}
else if (adc_value > 839 && adc_value <=891)
{
  temperatura = -0.191219 * adc_value + 210.410166;
}
else
{
  temperatura = -0.000007857923 * adc_value * adc_value - 0.17775 * adc_value + 204.6398;
}
```

FONTE: OS AUTORES (2018).

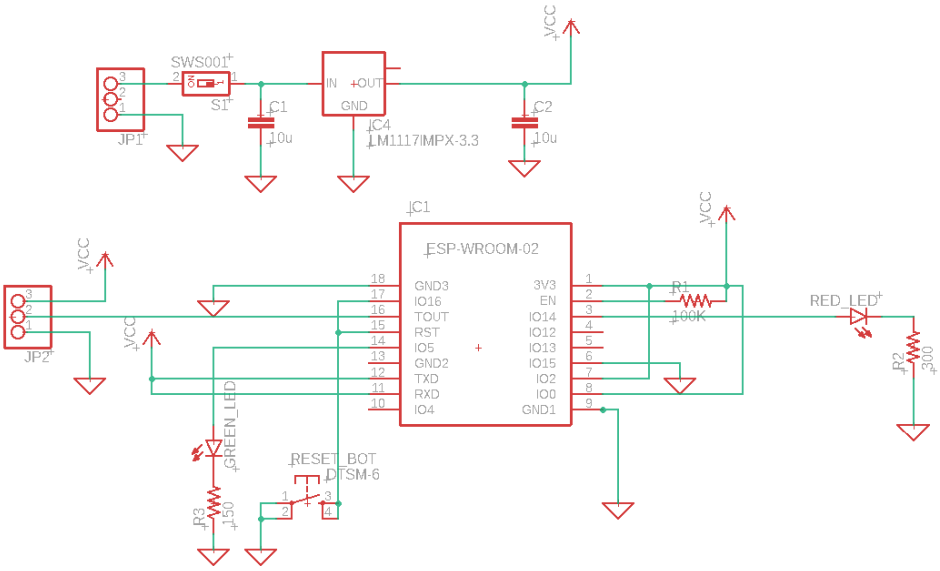
Depois da conversão, o valor da temperatura em decimal, é enviado para servidor ThingSpeak. Em todas vezes que o dado é enviado, a função LED_Verde é acionada, acendendo o LED verde, por um período de 50ms, para indicar o sucesso do envio.

Ainda dentro da função Loop, quando a rede não estiver conectada ou a conexão perdida, o *timer* – com duração de 50ms é ativado. Isto é: A cada 50ms, a

função LED_Verm é chamada para acionar o LED vermelho para chamar atenção do usuário, alertando-o para um problema com a conexão.

Após o desenvolvimento do código e os testes realizados no *protoboard*, próxima etapa é criação de circuito impresso, conforme desenhado (FIGURA 26).

FIGURA 26 – CIRCUITO ELÉTRICO FINAL DO TERMÔMETRO



FONTE: OS AUTORES (2018).

3.3.5. Programação direta no Chipset

Nesta etapa, utilizando o firmware desenvolvido na etapa anterior, muda-se o foco para a programação do ESP sem a utilização do kit de desenvolvimento. Assim, o objetivo desta fase era tornar possível programar um microcontrolador autônomo, desconectado de uma placa de desenvolvimento. Portanto, o objetivo desta fase é desenvolver um circuito em *protoboard* que torne possível a programação da unidade.

Assim, analisando o datasheet do ESP-WROOM-02, primeiramente seguiu-se o circuito mostrado pela fabricante (FIGURA 28).

Não sendo apresentada neste circuito a forma de alimentação do microcontrolador, foi buscado, dentro das informações fornecidas pela fabricante, o a montagem de um circuito em *protoboard*, munido de um regulador de tensão para 3,3V (FIGURA 27).

No entanto, tal diagrama do circuito fornece informações errôneas para a

programação do mesmo. De forma que o dispositivo não entra em modo de download, impossibilitando a transferência do firmware desenvolvido para o mesmo. Após pesquisa, foi possível encontrar outra página web referenciando uma diferente maneira de colocar o microcontrolador em modo de programação, de acordo com a conexão de alguns pinos do mesmo (TABELA 4).

Assim, foi elaborado outro circuito (FIGURA 29), juntando as informações do anterior (FIGURA 28) com o atual.

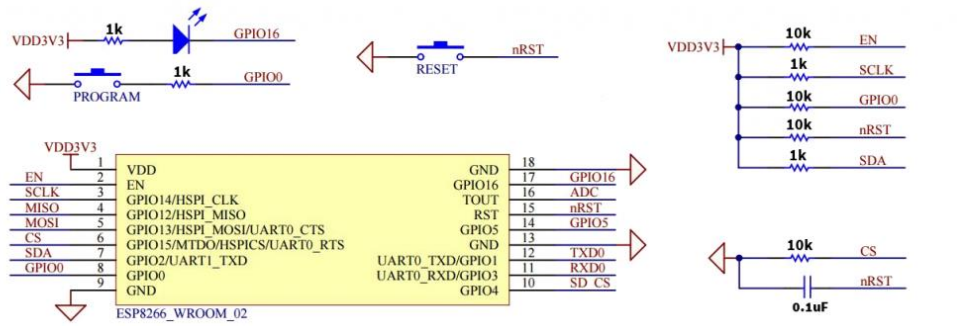
Em seguida, é compilado o firmware do código usando a IDE PlatformIO.

TABELA 4 – SEQUÊNCIA DE PINOS PARA MODOS DE BOOT

Modo	GPIO15	GPIO2	GPIO0
Modo de Escrita (Via UART)	GND	VCC	GND
Modo de Execução	GND	VCC	VCC
Modo de Execução com firmware do Cartão SD	VCC	GND	GND

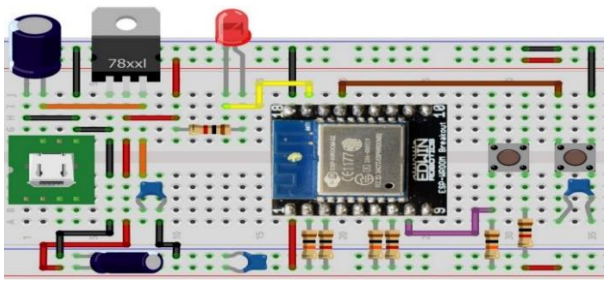
FONTE: OS AUTORES (2018)

FIGURA 27 – ESQUEMÁTICO PARA PROGRAMAR O ESP8266



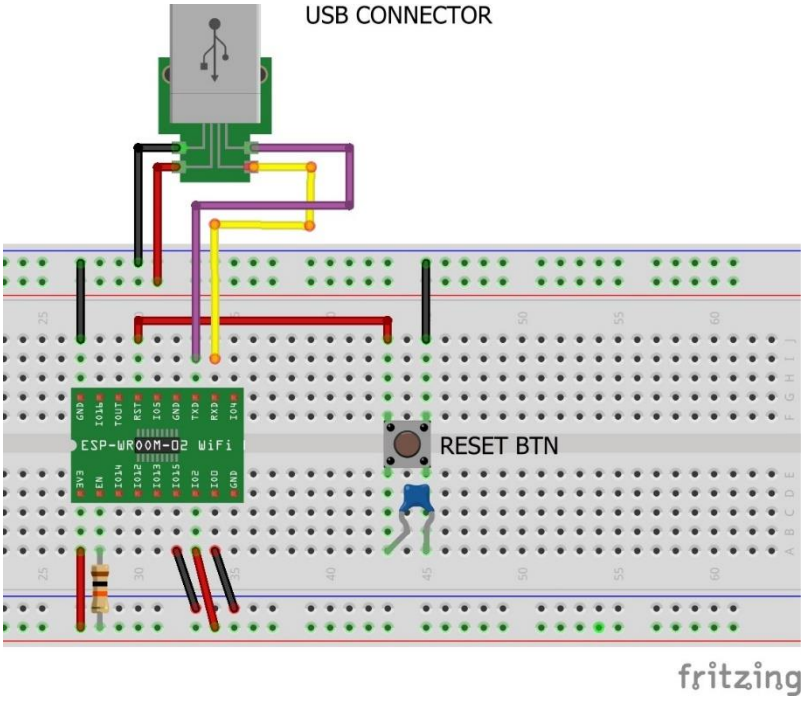
FONTE: EDWINROBOTICS (2018)

FIGURA 28 – *PROTOBOARD* PARA PROGRAMAR O ESP8266



FONTE: EDWINROBOTICS (2018)

FIGURA 29 – SEGUNDA PROTOBOARD PARA PROGRAMAR O ESP8266



FONTE: OS AUTORES (2018)

3.3.6. Android

Definida a topologia final, é necessário o desenvolvimento de uma aplicação em Android, para interface com a plataforma ThingSpeak e o bracelete. Separados em duas linhas de desenvolvimento, esta fase tratará apenas do desenvolvimento da aplicação *mobile*, enquanto outra detalhará o aperfeiçoamento do *firmware* para o ESP-WROOM-02.

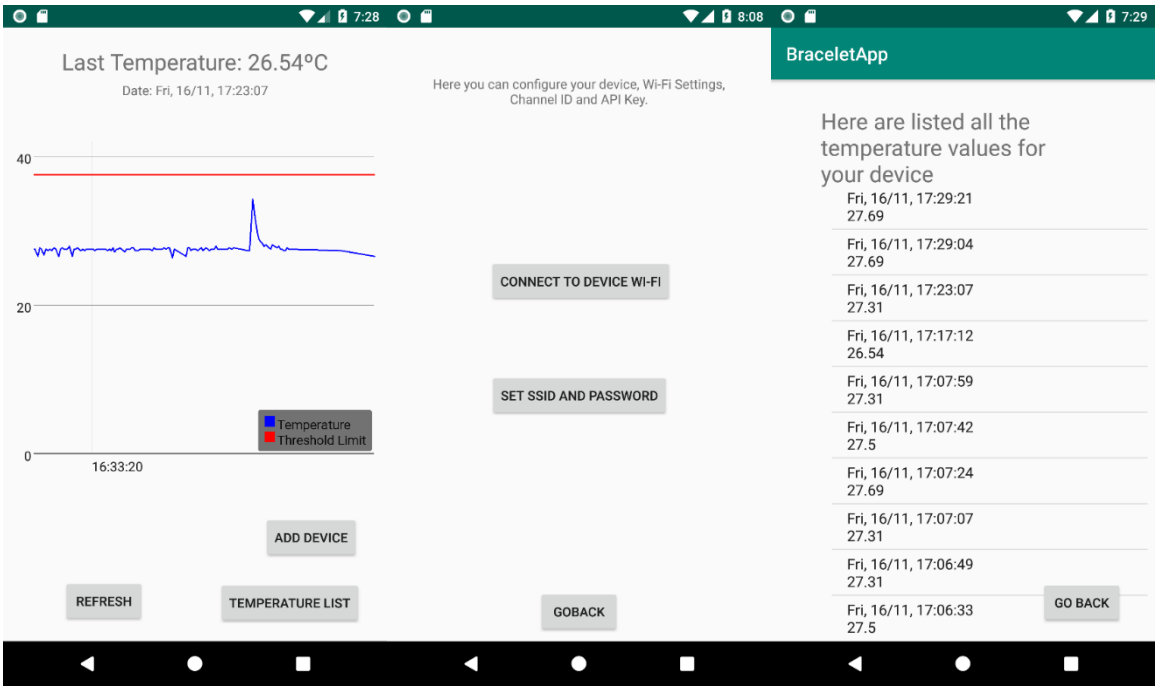
3.3.6.1. Interfaces

O aplicativo desenvolvido para comunicação com a plataforma ThingSpeak possui três telas distintas. Utilizando a IDE do *Android Studio*, foi possível desenvolver o código e definir as interfaces relatadas a seguir. Na tela principal (FIGURA 30), é demonstrado um gráfico com todos os valores de temperatura, e, na parte superior da tela é indicado ó último valor registrado de temperatura, com sua respectiva data e hora. Na parte inferior, botões de atualização e navegação as demais telas. O gráfico também é navegável e possui a função de zoom e mostra de valores quando clicados em um ponto específico.

A tela de configuração do bracelete (FIGURA 30) possui, no centro, dois botões para configuração do dispositivo: a primeira redireciona o aplicativo para a tela de redes Wi-Fi disponíveis, para que o usuário selecione primeiramente a rede interna do bracelete. Por segundo, ele seleciona o segundo botão, que abrirá diretamente a página HTTP para configuração do ESP para conectá-lo a uma rede sem fio.

A última tela (FIGURA 30) permite o usuário visualizar todos os valores de temperatura registrados desde a configuração do dispositivo, em uma lista, com os valores de data, hora e temperatura.

FIGURA 30 – INTERFACE PRINCIPAL DA APLICAÇÃO EM ANDROID



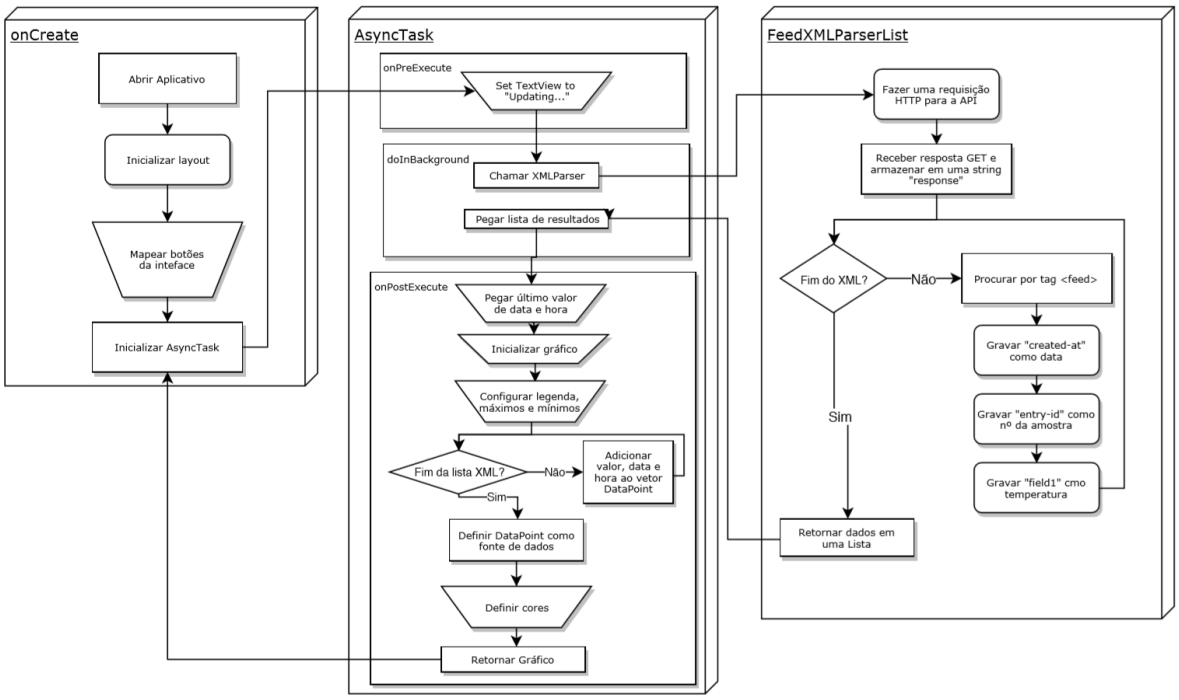
FONTE: OS AUTORES (2018).

Todos os algoritmos desenvolvidos nesta fase estão inclusos nos apêndices. (APÊNDICES 2-11). O diagrama de classes do código (FIGURA 32) explica a conexão entre as classes dentro da aplicação.

3.3.6.1. Desenvolvimento da aplicação

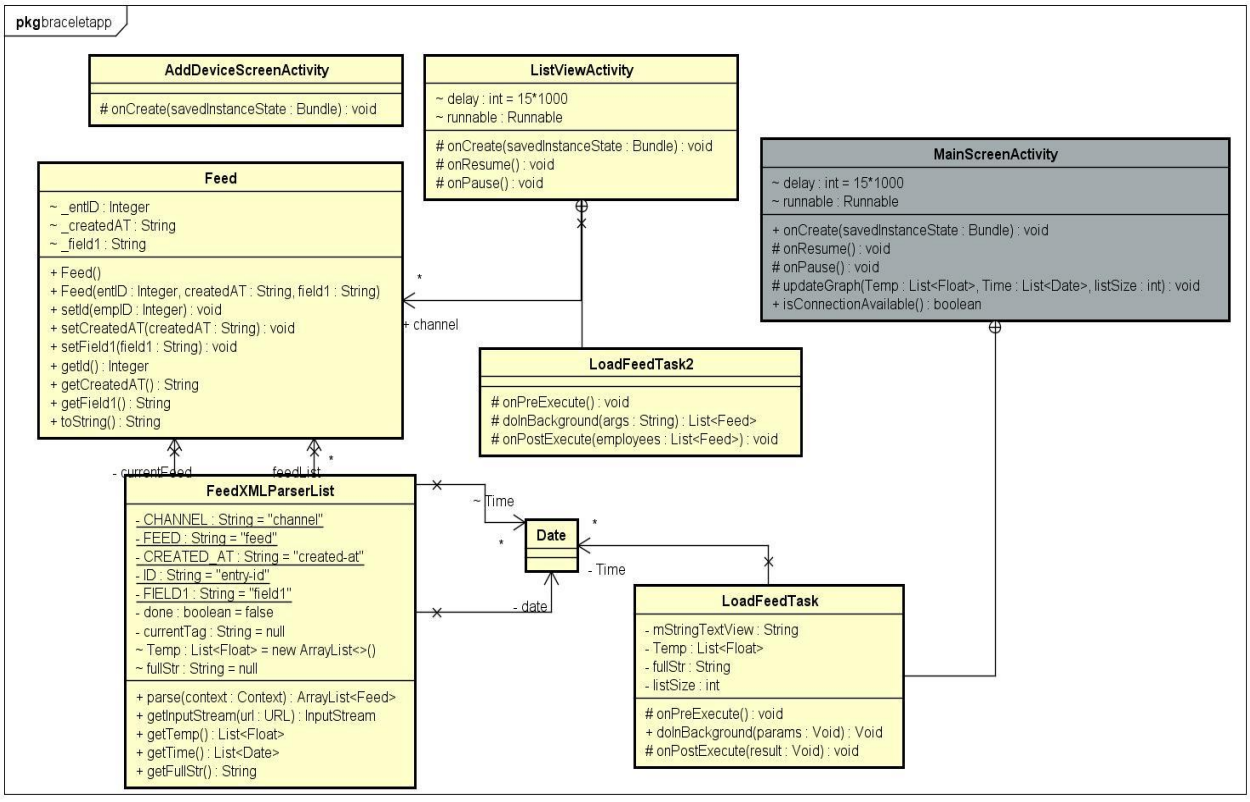
Desenvolvidas as interfaces, é necessário que a aplicação realize requisições e processe dados em segundo plano. O código (FIGURA 31) e suas classes (FIGURA 32x') serão detalhados a fundo nos próximos itens desta subseção.

FIGURA 31 – VISÃO GERAL DAS CLASSES DO APLICATIVO



FONTE: OS AUTORES (2018).

FIGURA 32 – DIAGRAMA DE CLASSES DA APLICAÇÃO ANDROID



FONTE: OS AUTORES (2018).

3.3.6.2. Comunicação com o ThingSpeak

A base para todo o funcionamento deste projeto consiste na comunicação entre a plataforma ThingSpeak, o dispositivo e a aplicação em Android, pois é necessário o registro dos dados na plataforma para que estes sejam demonstrados no aplicativo.

Portanto, a aplicação em Android utilizará a API da plataforma para obter os dados em XML, através de uma requisição GET, e assim analisará os dados através de uma classe desenvolvida especificamente para esta aplicação (FIGURA 34).

Iniciada a classe, o aplicativo realizará a requisição “GET” da URL, que retornará como resposta um XML contendo os dados publicados no canal dentro da plataforma (FIGURA 33).

Em seguida, o aplicativo percorerrá por inteiro o arquivo XML, e classificará cada item por sua data e valor de temperatura, retornando para a classe principal uma variável do tipo Lista.

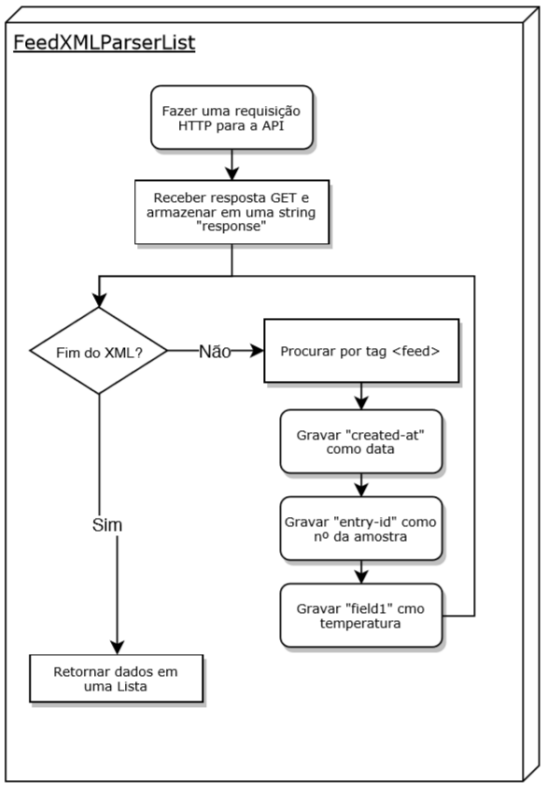
FIGURA 33 – EXEMPLO DE DADOS RETORNADOS PELA API

https://api.thingspeak.com/channels/489402/feeds.xml

```
▼<channel>
  <id type="integer">489402</id>
  <name>NodeMCU PlatformIO</name>
  <latitude type="decimal">0.0</latitude>
  <longitude type="decimal">0.0</longitude>
  <field1>Temperature</field1>
  <created-at type="dateTime">2018-05-05T14:36:44Z</created-at>
  <updated-at type="dateTime">2018-11-14T22:22:01Z</updated-at>
  <last-entry-id type="integer">1379</last-entry-id>
 ▼<feeds type="array">
  ▼<feed>
     <created-at type="dateTime">2018-11-17T19:46:46Z</created-at>
     <entry-id type="integer">1280</entry-id>
     <field1>26.73 </field1>
   </feed>
  ▼<feed>
     <created-at type="dateTime">2018-11-17T19:47:03Z</created-at>
     <entry-id type="integer">1281</entry-id>
     <field1>26.73 </field1>
   </feed>
  ▼<feed>
     <created-at type="dateTime">2018-11-17T19:47:22Z</created-at>
     <entry-id type="integer">1282</entry-id>
     <field1>27.11 </field1>
   </feed>
```

FONTE: OS AUTORES (2018).

FIGURA 34 – FUNÇÃO PARA ANÁLISE DOS DADOS EM XML



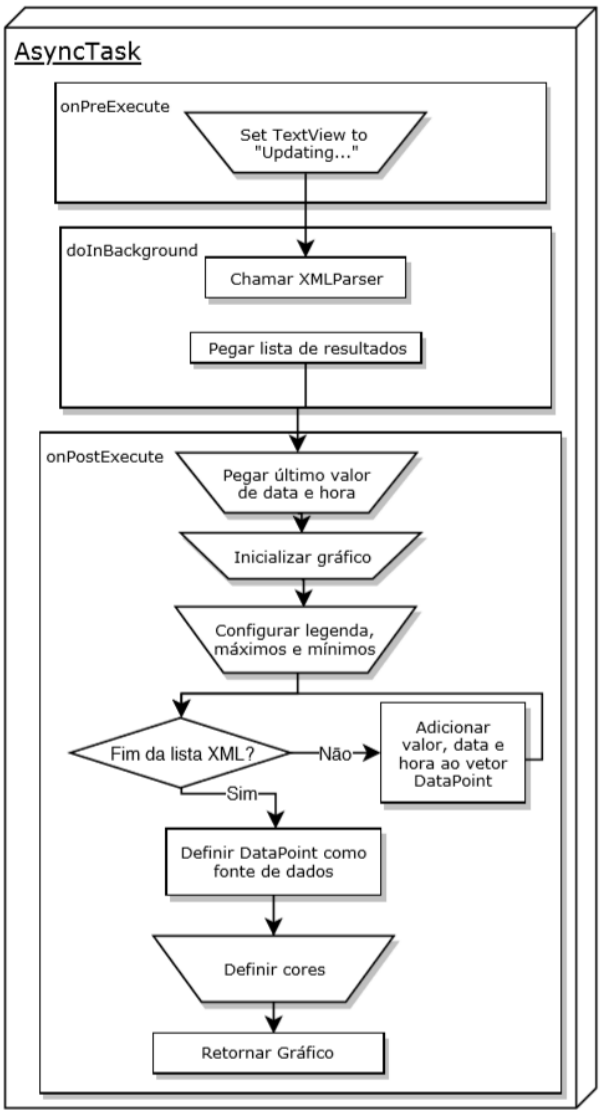
FONTE: OS AUTORES (2018).

3.3.6.3. AsyncTask

Para manipulação dos dados retornados nesta pela classe que analisa os dados em XML, é necessário utilizar a classe abstrata AsyncTask do Android, permitindo que operações sejam executadas em segundo plano, alterando seguramente a interface com o usuário após o fim da operação, sem a necessidade da manipulação de *threads* ou *handlers*. (ANDROID, 2018)

Esta classe ajudante define uma tarefa assíncrona para o sistema, e é separada em 3 subfunções: *onPreExecute, doInBackground e onPostExecute* (FIGURA 35).

FIGURA 35 – FLUXOGRAMA DA CLASSE ASYNCTASK



FONTE: OS AUTORES (2018).

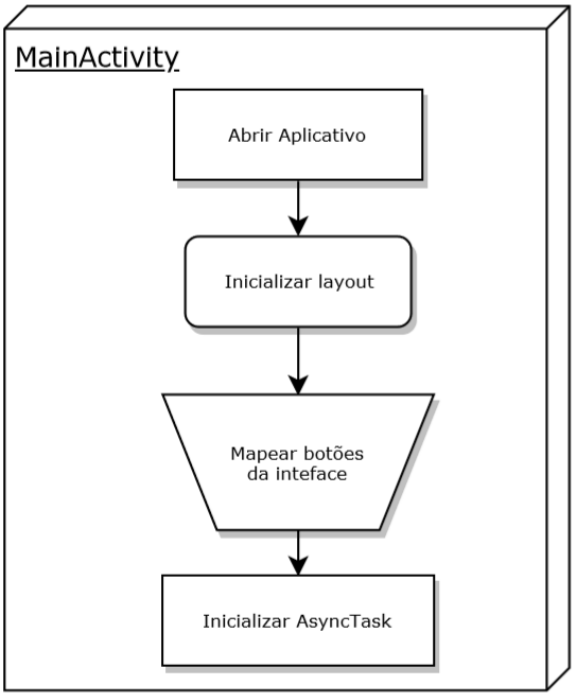
Para esta aplicação, estas sub funções são utilizadas para receber a lista retornada pela outra classe, e atualizar os dados na interface do usuário.

Assim, cada vez em que o botão “Atualizar” é pressionado, esta atividade será executada em segundo plano, para aquisição dos dados e atualização dos campos da interface.

3.3.6.4. MainActivity

Para a classe principal, apenas foram referenciados os botões da interface com os *layouts* respectivos, de forma a manter organizado o desenvolvimento desta aplicação (FIGURA 36).

FIGURA 36 – FLUXOGRAMA DA CLASSE MAINACTIVITY

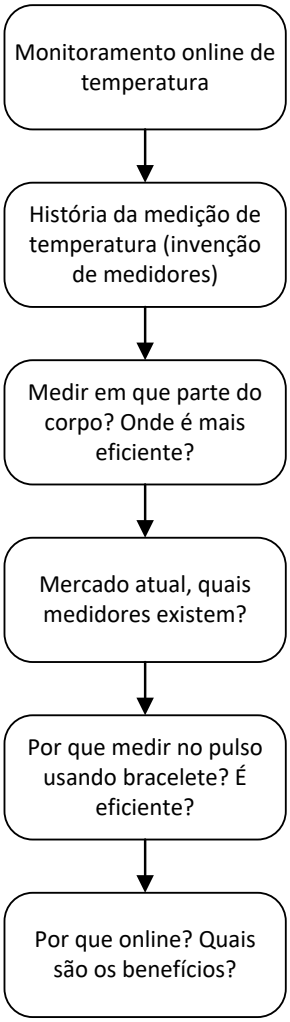


FONTE: OS AUTORES (2018).

4. RESULTADOS E DISCUSSÃO

Inicialmente, a pesquisa ocorreu de maneira razoavelmente satisfatória, mas não eficiente. Pois, por exemplo, ao pesquisar sobre monitoramento *online* de temperatura, gastou-se muito tempo na pesquisa sobre o exato assunto. O que ocasionou uma enorme quantidade de informação que precisa ser filtrada ou selecionada de alguma forma. Ou seja, o trabalho estava sem direção para alcançar o resultado. A eficiência na pesquisa, portanto, era baixa até elaborar um roteiro em uma folha de papel, onde constavam os passos da pesquisa. Desde então, a mesma tornou-se mais proveitosa (FIGURA 37).

FIGURA 37 - FLUXOGRAMA - MONITORAMENTO DE TEMPERATURA



FONTE: Os autores (2016).

Para a etapa de desenvolvimento de código, implementação de hardware e

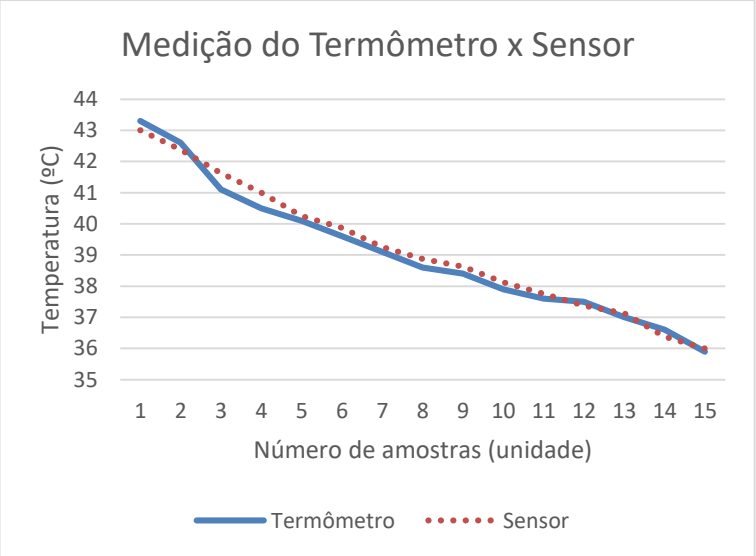
tecnologias utilizadas, foram desenvolvidas as seguintes conclusões, de acordo com as secções realizadas na etapa de desenvolvimento.

Na etapa de projetos preliminares, a utilização do módulo HM-10 juntamente com o microcontrolador MSP430 agregou conhecimentos para o desenvolvimento, pois, inicialmente, o aplicativo do *smartphone* recebia uma letra por vez, ao invés de uma *string* por inteiro. Foi necessária a implementação de uma lógica para que a interrupção do TIMER só era ativada depois de ter enviada a *string* inteira (FIGURA 13).

Após os estudos iniciais e a revisão mais detalhada da bibliografia, verificou-se que o transdutor de temperatura utilizado não atendia as necessidades do projeto, e o protocolo 6LoWPAN desejado apenas poderia ser utilizado no CC2650 desde que uma *stack* fosse comprada. Desta maneira, substitui-se o DS18B20 pelo sensor de temperatura LMT70, por sua exatidão garantida de 0,05ºC na faixa de 20ºC a 40ºC, alimentação nominal de 2 a 5,5 Volts e faixa de medição de -55 a +150°C (TEXAS INSTRUMENTS, 2017).

Por segundo, a utilização do Raspberry Pi como servidor apenas local não atenderia a parte de solução IoT, devido a necessidade de um Raspberry Pi para cada sensor (aumentando o custo final do produto).

FIGURA 38 - GRÁFICO TERMÔMETRO X SENSOR DS18B20

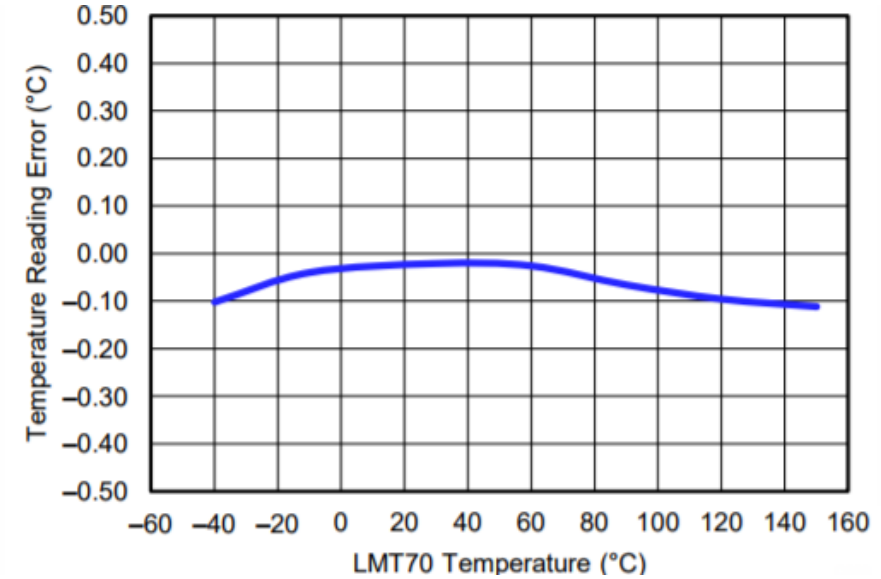


FONTE: Os Autores (2017).

Como pode observar, o gráfico do erro de leitura de temperatura em função do

valor da temperatura (FIGURA 39) mostra que na faixa de 20ºC a 40ºC, o sensor LMT70 apresenta o menor erro de leitura. Isso é perfeito para presente projeto, em que a precisão da medição é essencial. Visto que é uma aplicação na área médica. Com este resultado de análise de informação fornecida pelo fabricante, a escolha do sensor é definitiva para este projeto.

FIGURA 39 - ERRO DE LEITURA EM FUNÇÃO DA TEMPERATURA



FONTE: Texas Instruments (2017).

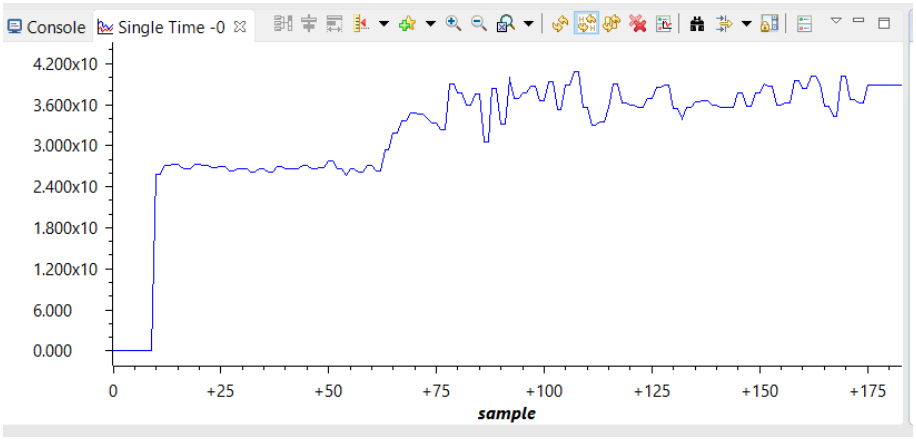
Com a adição do LMT70 ao projeto, foi necessária a consulta ao *datasheet* do fabricante do sensor. Percebeu-se que o sensor, ao ser alimentado, ele mede a temperatura onde ele se encontra e, através do conversor AD próprio, converte o resultado em mV. E para obter a temperatura resultante da medição em grau Celsius, é preciso usar uma função de transferência da saída, fornecida pelo fabricante no *datasheet.*

O resultado imediato (FIGURA 40), que exibe uma conversão não estável do valor de temperatura. Para corrigir este problema, foi implementado um método na programação para calcular a média móvel do valor lido. E o resultado mostra que a variação é muito mais suave (FIGURA 41).

Para implementar o sistema que possa ler o sensor, foi necessário usar função ADC do CC2650. Apesar de não há muita referência na comunidade do TI, por ser um kit de lançamento recente. Mas através da biblioteca disponibilizada pela TI, avançou-se a implementação da mesma. Além disso, a aprendizagem sobre UART também foi

necessário pois o futuro trabalho depende deste meio para enviar os dados, e posteriormente usando-os para resumir em um gráfico em que todos usuários cadastrados possam acessá-lo caso queiram.

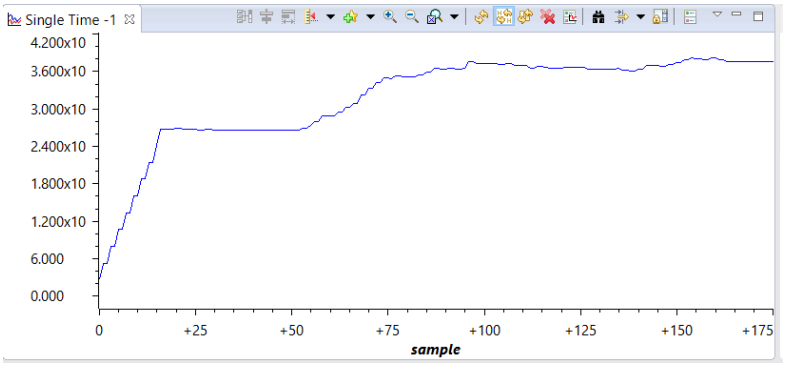
FIGURA 40 - RESULTADO SEM A MÉDIA MÓVEL



FONTE: Os Autores (2017).

Para verificação da eficiência da conversão ADC do sensor LMT70 fornecida pela Texas Instruments, efetua-se um teste com o ESP8266 e a plataforma ThingSpeak para averiguar a qualidade dos dados enviados para esta última. Assim, durante um período de 3 minutos, a temperatura ambiente com ar condicionado ajustado para aproximadamente 23ºC, deixa-se o sensor em cima de uma superfície. Em seguida, segura-se o sensor na ponta dos dedos por 5 minutos, e verifica-se uma elevação nos valores de temperatura enviados para o serviço ThingSpeak. Por último, coloca-se o sensor na mesma localização em que estava, de forma a observar a estabilização do mesmo em temperatura ambiente (FIGURA 42).

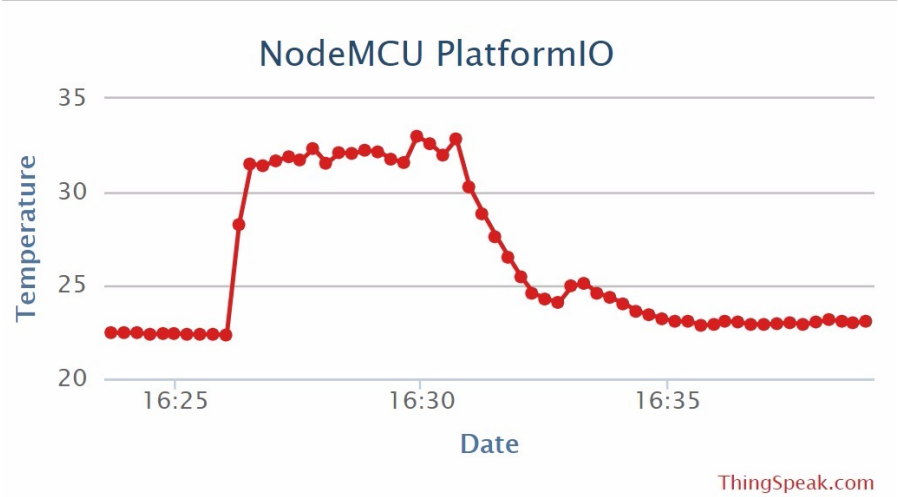
FIGURA 41 - RESULTADO COM A MÉDIA MÓVEL



FONTE: Os Autores (2017).

Na etapa de programação do Microcontrolador, a dificuldade para programar um ESP8266 autônomo provou-se ser extremamente maior do que a esperada, devido a configuração nos quais certos pinos de IO devem ser conectados em locais específicos. Sendo assim, no semestre de TCC-B, um aplicativo móvel para se comunicar com este microcontrolador será desenvolvido para elevar a complexidade do projeto.

FIGURA 42 - MEDIÇÃO DE TEMPERATURA ENVIADA AO THINGSPEAK



FONTE: os autores (2018).

Durante o teste, a troca de nome do WiFi, a modificação da senha de acesso para WiFi, ou até mesmo o desligamento e depois religa-lo, todos esses eventos retornaram resultados esperados.

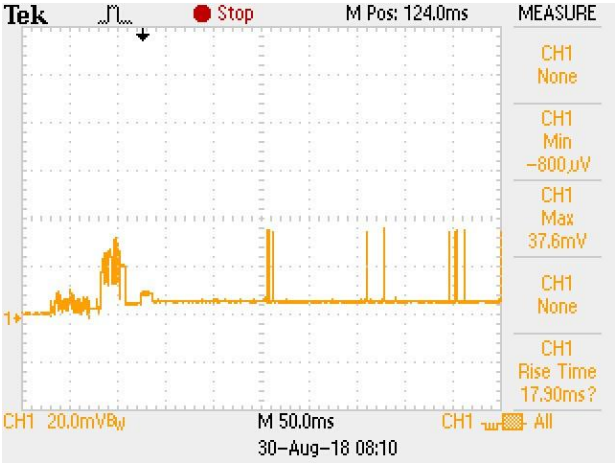
Em relação ao sensor LMT70, de acordo com o *datasheet*, fornecido pelo fabricante do mesmo, o sensor é ideal para aplicação medica como termômetro e que ao ser usado na faixa de 30 a 45ºC, o funcionamento é linear e, portanto, despensa a necessidade de calibração.

Para escolher uma bateria adequada para alimentar o circuito, é fundamental saber o consumo do mesmo para que depois a realização dela. Para isso, o circuito é ligado com uma fonte externa com capacidade de fornecer até um ampere de corrente. E um resistor um ohm é colocado em serie com a entrada do circuito de tal forma que pode ser medido a forma de corrente do mesmo com o osciloscópio.

Em relação a alimentação, foi encontrada dificuldade para empregar a bateria de tamanho adequado e que consiga fornecer uma corrente constante mínima de

100mA. Depois de muita pesquisa, a bateria mais adequada para presente projeto é uma bateria de Íons de Lítio 200mAh cuja tensão máxima é de 3,7V. Mas a tensão de bateria (3,7V) não pode ser aplicada diretamente no ESP (segundo fabricante, faixa de operação com tensão de entrada de 2,5V a 3,6 V). Para evitar que o ESP seja alimentado com 3.7V, um diodo é aplicado em serie com a bateria de tal forma a diminuir a tensão aplicada para 3v. Portanto, a solução encontrada para esse problema é usar duas baterias em série (7,4V) e então usar regulador de tensão (LM1117) para fornecer uma tensão constante de 3,3V. A eficácia dessa associação de bateria demonstrou, na montagem do circuito no *protoboard*, o resultado esperado, isto é, usando duas baterias e regulador de tensão fez com que o ESP consiga ler corretamente a tensão fornecido pelo sensor de temperatura (LMT70).

FIGURA 43 - FORMA DE ONDA DE CORRENTE DO ESP8266



FONTE: os autores (2018).

Outra dificuldade é usar um servidor que consiga receber as temperaturas medidas em maior frequência, isto é, ele consiga receber uma amostra por segundo, e mínima de uma amostra por minuto. Mas por ser um servidor com serviço gratuito e que proposito de teste funcional do protótipo, o uso do servidor ThingSpeak é definido como uso temporário e que no futuro, caso necessário, usar outro servidor pago.

Foi percebido de que o microcontrolador possui um problema na amostragem e quantização. Isto é, devido ao microcontrolador possui 10 *bits.* Isto é, na temperatura ambiente do local onde encontrava o circuito, a tensão de saída do sensor, usando multímetro para medir, era 0,932V, mas a tensão lida pelo ESP, na saída do monitor serial mostrava 1024. Por ter uma conversão de 10 bits, o valor de 1024 equivale uma

tensão de 1V, mas na verdade era fornecida 0,932V. Isso significa que o ESP, por ter uma única entrada de conversão analógica-digital e que a tensão de entrada estar limitada em 0 a 1V, não conseguiu, de alguma forma, ler corretamente a tensão que LMT70 fornece.

Uma solução para esse problema foi que, ao aproximar gradativamente ferro de solda ligado ao sensor, a tensão dele, medida via multímetro, caia e a valor binário do ESP também caia proporcionalmente de tal forma que a razão entre o valor mostrado pelo ESP e tensão medida do sensor é sempre quase a mesma, sendo que ela é aproximadamente 1,06217. Com isso, este multiplicador é implementado no código do ESP para que a correção seja feita.

Um protótipo foi desenvolvido e ele simplesmente não funcionou. Ao investigar o motivo, foi constatado de que os pinos do TX e do RX, quando o modulo ESP em operação, ambos pinos têm que estar conectados ao VCC.

Em relação a aplicação desenvolvida para *Android*, foi necessária a implementação do elemento *GraphView* a aplicação, e reorganização da interface severas vezes, de forma a torna-la mais intuitiva e amigável ao usuário. A dificuldade encontrada para obtenção de dados em segundo plano provou-se difícil, por ser executada em segundo plano. Assim, após busca e pesquisa por uma solução funcional, foi implementado na aplicação o AsyncTask, otimizando o uso de processos e *threads* executadas em segundo plano, sem necessidade de agendar processos para serem executados periodicamente.

5. CONCLUSÃO

Na elaboração deste trabalho, surgiram muitas dúvidas, alguns acertos e vários erros. No início, levou-se mais de um mês para escolher um bom projeto, o que ocasionou uma demora para iniciar o trabalho. Pois haviam dúvidas sobre para qual área de aplicação que o projeto seria direcionado, se para área acadêmica, para área industrial, para área da saúde ou para área do entretenimento. Também surgiram preocupações com a aplicabilidade do projeto, a sua possível aceitação comercial e as possíveis dificuldades durante a implementação, entre outras. Sendo assim, ocorreu também a demora para a escolha de um professor orientador. Pois cada professor tem sua especialidade, dependendo da sua área de formação e experiência profissional.

O desenvolvimento deste projeto, possibilitou o reestudo e aprofundamento no conhecimento na área de eletrônica, além de aprender produzir, ainda de forma rudimentar, placa de circuito eletrônico impresso, com componente SMD. Ao fazer testes do circuito no protoboard e a programação do código do dispositivo, surgiram várias dificuldades e aos poucos, sanados, uma depois da outra. Uma dela foi o encapsulamento do sensor que exigiu o estudo de características de condução elétrica de cada material e escolher o melhor deles. Além disso, a realização de pesquisa sobre vários tipos de micro controladores presentes no mercado permitiu se atualizar no mercado de tecnologia. Assim como conhecer, mesmo elementar, um pouco de página em HTML, linguagem C++, comunicação sem fio. Nesse sentido, o aprendizado satisfatório e os recursos mais usados foram livros técnicos relacionados e trabalhos de tese publicados. Isto é, aprender ser autodidata foi um dos objetivos proposto nesse presente trabalho.

Em relação a aplicação desenvolvida para *Android*, devido ao enorme volume de desenvolvedores *Android* no mercado, foi possível encontrar referências para auxílio no desenvolvimento do aplicativo, desde a parte para organização do *layout* da aplicação até a implementação do recurso de gráfico e navegação para outras telas, o que se provou complexo, devido ao nível de abstração presente na linguagem de programação Java. Sendo assim, para criação da interface gráfica, um estudo a fundo nas referências citadas neste documento foi necessário para adequado desenvolvimento e funcionalidade da mesma.

Quanto a sua funcionalidade, a implementação de uma classe assíncrona para

atualização dos dados em segundo plano foi descoberta de forma empírica, devido à testes realizados em versões de desenvolvimento, e notado o travamento da interface gráfica durante a obtenção dos dados da API ThingSpeak. Por ser uma plataforma concebida visando facilitar o uso de dispositivos IoT, o serviço da MathWorks provou-se facilitador e funcional para o desenvolvimento deste projeto como um todo, armazenando seguramente os dados e disponibilizando-os de forma adequada para a aplicação em *Android*. Novamente, graças a comunidade de desenvolvedores, problemas foram solucionados devido a um estudo nas referências citadas.

5.1. TRABALHOS FUTUROS

Devido ao prazo para apresentação, não foi possível concluir o protótipo cabeça-de-série, mas apenas um protótipo funcional. Para transformá-lo num produto, deverão executados os trabalhos futuros relacionados abaixo:

1. Substituição do regulador de tensão pelo conversor buck-boost TPS63036;
2. Projeto da PCI final;
3. Design do bracelete;
4. Continuidade do aplicativo;
5. Montagem do cabeça-de-série;
6. Ensaios em pacientes com aprovação preliminar do comitê de ética médica.

7. REFERÊNCIAS BIBLIOGRÁFICAS

TOWNSEND, Kevin. ***Getting started with Bluetooth Low Energy****.* O'Reilly, 2014.

VERMSAN, O. ***Internet of Things – From Research and Innovation to Market Deployment.*** River Publishers, Aalborg, 2014.

CARVALHOSA, M. C. **Técnicas Laboratoriais de Física - Bloco I**. Porto, 1994.

ELKIN, M. **Intervenções de Enfermagem e Procedimentos Clínicos.** Lusociência, 2000.

FUKUSHIMA, Y. ID-based Communications for Future IoT/M2M**.** ***Journal of the National Institute of Information and Communications Technology.*** 2015. Vol.62 No. 2.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 6023**: referências bibliográficas. Rio de Janeiro, 1989.

TABISH, Rohan et al. **"*A comparative analysis of BLE and 6LoWPAN for U-HealthCare applications*"**. Em: *GCC Conference and Exhibition* (GCC), 2013 7th IEEE. IEEE, 2013. p. 286-291.

JOHANN, S. ***"Multiprocessor System on a Chip"***. Rio Grande do Sul, 2014. Artigo – PUC-RS

ZAMBARDA, P. **"Internet das Coisas: entenda o conceito e o que muda com a tecnologia".** TechTudo. Disponível em: http://www.techtudo.com.br/noticias/noticia/2014/08/internet-das-coisas-entenda-o-conceito-e-o-que-muda-com-tecnologia.html. Acesso em: (17 set. 2016)

ERICSSON**.** ***"More than 50 billion connected devices".*** White Paper, fevereiro de 2011". Disponível em: http://www.akos-rs.si/files/Telekomunikacije/Digitalna_agenda/Internetni_protokol_Ipv6/More-than-50-billion-connected-devices.pdf. Acesso em: (17 set. 2016)

HIGH, P. *"**Top 10 Strategic Technology Trends for 2014"**.* Disponível em: http://www.forbes.com/sites/peterhigh/2013/10/14/gartner-top-10-strategic-technology-trends-for-2014/#. Acesso em: (18 set. 2016)

ACTFORLIBRARIES. **"*The History of the Body Temperature Thermometer*"**. Disponível em: http://www.actforlibraries.org/the-history-of-the-body-temperature-thermometer/ Acesso em: (18 set. 2016)

ANDERSON, E. "Factors Influencing the Body Temperature of 3-4-month-old infants

at home during the day". ***U. S. National Library of Medicine.*** 1990. Vol. 65 No. 12.

HEALTHYCHILDREN. ***"Fever and Your Baby".*** Disponível em: https://www.healthychildren.org/English/health-issues/conditions/fever/Pages/Fever-and-Your-Baby.aspx. Acesso em: (18 set. 2016)

BABYEARTH. ***"Baby Temperature Range: What's Normal?"*** Disponível em: http://www.babyearth.com/grow/baby-temperature-range-whats-normal/ Acesso em: (18 set. 2016)

BABYCENTER. ***"Fever in babies"***. Disponível em: http://www.babycenter.com/0_fever-in-babies_84.bc?showAll=true Acesso em: (18 set. 2016)

GIANG, H. ***"How to Communicate Module Bluetooth HM-10 (4.0) With Android Smartphone and MCU"***. Disponível em: http://www.instructables.com/id/How-to-Communicate-Module-Bluetooth-HM-10-40-With-/?ALLSTEPS Acesso em: (18 set. 2016)

CHEN, D. ***"How to use Bluetooth 4.0 HM10"***. Disponível em: http://www.instructables.com/id/How-to-Use-Bluetooth-40-HM10/ Acesso em: (18 set. 2016).

TEXAS INSTRUMENTS. ***"SimpleLink Bluetooth Low Energy and Proprietary wireless MCU"***. Disponível em: http://www.ti.com/product/CC2541/technicaldocuments Acesso em: (18 set. 2016).

LANGER, F. Ways to measure body temperature in the field. ***Journal of Thermal Biology.*** 2014. Vol. 42 No. 46.

ASCOM. **"Termômetros com mercúrio podem ser proibidos"**. Disponível em: http://portal.anvisa.gov.br/noticias/-/asset_publisher/FXrpx9qY7FbU/content/termometros-com-mercurio-podem-ser-proibidos/219201 Acesso em: (18 set. 2016).

DUARTE, N. **"Dispositivos SoC da Altera – desempenho para projetos embarcados".** Disponível em: http://www.embarcados.com.br/dispositivos-soc-da-altera-alto-desempenho-para-projetos-embarcados/ Acesso em: (18 set. 2016).

TIMES, E. ***"The Great Debate: SoC vs. SIP"***. Disponível em: http://www.eetimes.com/document.asp?doc_id=1153043. Acesso em: (18 set. 2016).

BLUETOOTH SIG. ***"Bluetooth Core Specification"***. Disponível em:

https://www.bluetooth.com/specifications/bluetooth-core-specification. Acesso em: (22 set. 2016)

ASHTON, K. ***"That 'Internet of Things' Thing".*** Disponível em: http://www.rfidjournal.com/articles/view?4986. Acesso em: (22 set. 2016)

BOLUTEK. "***AT COMMANDS for BLE-CC41-A Bluetooth Mobile".*** Disponível em: http://www.hangar42.nl/wp-content/uploads/2015/08/CC41-at-commands.pdf. Acesso em: (06 out. 2016).

LOVETTE, K. ***"BLE View App for iPhone".*** Disponível em: http://ble-view.appstor.io/. Acesso em: (06 out. 2016).

MAXIM INTEGRATED. ***"DS18B20 Programmable Resolution 1-Wire Digital Thermometer".*** Disponível em: http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf. Acesso em: (10 out. 2016)

KURUP, R. ***"DS18B20 Interfacing"***. Disponível em: https://www.embeddedrelated.com/showcode/294.php. Acesso em: (11 out. 2016)

MULLIGAN, G. ***"The 6LoWPAN architecture"*** Em: *Proceedings of the 4th workshop on Embedded networked sensors*. ACM, 2007. p. 78-82.

NIC.BR. **"Introdução ao IPv6"**. Disponível em: https://ipv6.br/post/introducao. Acesso em: (30 mar. 2017)

HAGUE, A. et al. ***"Raspberry Pi Education Manual"***. Raspberry Pi Foundation, 2012.

TEXAS INSTRUMENTS. "**CC2640 and CC2650 SimpleLink™. Bluetooth. ® low energy Software Stack 2.2.1 Developer's Guide**". Disponível em: http://www.ti.com/lit/swru393. Acessado em: (20 mai. 2017).

OTTO, M., THORNTON, J. ***"Boostrap front-end framework".*** Massachussets Institute of Technology, 2010. Disponível em: http://getbootstrap.com/. Acesso em: 30 mai. 2017).

JINJA2. ***"Welcome to Jinja2"***. Disponível em: http://jinja.pocoo.org/docs/2.9/. Acesso em: (10 out. 2017)

MATERIALDESIGN. ***"Introduction – Material Design"***. Disponível em: https://material.io/guidelines/#. Acesso em: (11 out. 2017)

NODEBR. ***“NodeBR – JavaScript Platform Framework”***. Disponível em: http://nodebr.com/. Acesso em: (11 out. 2017)

EDWINROBOTICS. **“*Getting Started with ESP-WROOM-02*”**. Disponível em: http://learn.edwinrobotics.com/getting-started-with-esp-wroom-02/ Acesso em: (18 abr. 2018).

ANDROID DEVELOPERS. **“AsyncTask”**. Disponível em: https://developer.android.com/reference/android/os/AsyncTask. Acesso em: (18 out. 2018).

8. APÊNDICES

APÊNDICE 1 - CÓDIGO PARA O ESP8266

```
#include <ESP8266Wifi.h>

//Definir o SSID da rede WiFi
const char* ssid = "IPHONE";
//Definir a senha da rede WiFi
const char* password = "erik1234";
float delta = 0;
float tempBuffer[10];
float temperatura = 0;
float temperaturaMedia = 0;
float temperaturaMediaOld = 0;
int posicao;
int i;
float adc_value = 0;

//Colocar a API Key para escrita neste campo
//Ela é fornecida no canal que foi criado na aba API Keys
String apiKey = "6PO28SIFTD5MV65A";
const char* server = "api.thingspeak.com";

WiFiClient client;

void setup() {
  //Inicia o WiFi
  WiFi.begin(ssid, password);

  //Espera a conexão no router
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}

void loop() {
```

```
  //Leitura de temperatura
  const float aref = 2.56;  // adjust it to the actual AREF voltage

  float adc_Value = 0;

  //AD converter
  adc_value = analogRead(A0);

  if (adc_value >=891 && adc_value <=943) //de 30ºC a 40ºC
  {
      temperatura = -0.19212665 * adc_value + 211.2190437;
  }

  else if (adc_value >= 839 && adc_value <=891) //de 40ºC a 50ºC
  {
      temperatura = -0.191219 * adc_value + 210.410166;
  }

  else
  {
      temperatura = -0.000007857923 * adc_value * adc_value - 0.17775 *
adc_value + 204.6398;//função de transferencia de degundo graus
  }

    if (posicao < 10)
    {
        tempBuffer[posicao] = temperatura;
        posicao++;
    }

    else
    {
        posicao = 0;
    }

    for (int i=0; i<10; i++)
    {
       temperatura = temperatura + tempBuffer[i];
```

```
  }

  temperaturaMediaOld = temperatura*0.76/10;

//Se não for um numero retorna erro de leitura
if (isnan(temperatura)) {
  Serial.println("Erro ao ler o sensor!");
  return;
}

//Inicia um client TCP para o envio dos dados
if (client.connect(server,80)) {
  String postStr = apiKey;
         postStr +="&amp;field1=";
         postStr += String(temperaturaMedia);
         postStr += "\r\n";

   client.print("POST /update HTTP/1.1\n");
   client.print("Host: api.thingspeak.com\n");
   client.print("Connection: close\n");
   client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
   client.print("Content-Type: application/x-www-form-urlencoded\n");
   client.print("Content-Length: ");
   client.print(postStr.length());
   client.print("\n\n");
   client.print(postStr);

delta = (temperaturaMedia - temperaturaMediaOld)/temperaturaMediaOld;

if (delta <= 0.5) {
  delay(500);
}

else if (delta>0.5 && delta<= 1) {
  delay(700);
}

else {
```

```
    delay(1000);
  }

  temperaturaMedia = temperaturaMediaOld;
  temperaturaMedia = temperaturaMedia*2;
  client.stop();
}
}
```

APÊNDICE 2 – ANDROIDMANIFEST.XML

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.erikakiyama.braceletapp">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
/>
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity
            android:name=".MainScreenActivity"
            android:label="@string/app_name"
            android:theme="@style/AppTheme.NoActionBar"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <activity
            android:name=".AddDeviceScreenActivity"
            android:theme="@style/AppTheme.NoActionBar"
            android:screenOrientation="portrait" />
        <activity android:name=".ListViewActivity"></activity>
    </application>

</manifest>
```

APÊNDICE 3 – ADDDEVICESCREENACTIVITY.JAVA

```
package com.example.erikakiyama.braceletapp;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.provider.Settings;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;

public class AddDeviceScreenActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.add_device_screen);

        Button addDev= findViewById(R.id.MainScreen_btn);
        addDev.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                startActivity(new
Intent(getApplicationContext(),MainScreenActivity.class));
            }
        });

        Button WifiOpen= findViewById(R.id.WifiSelScrn_btn);
        WifiOpen.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                startActivity(new Intent(Settings.ACTION_WIFI_SETTINGS));
            }
        });

        Button ChromeOpen= findViewById(R.id.BrowserOpen_btn);
```

```
        ChromeOpen.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                String url = "http://192.168.4.1";
                Intent i = new Intent(Intent.ACTION_VIEW);
                i.setData(Uri.parse(url));
                startActivity(i);
            }
        });

    }
}
```

APÊNDICE 4 – FEED.JAVA

```
package com.example.erikakiyama.braceletapp;


public class Feed {
    Integer _entID;
    String _createdAT;
    String _field1;

    // constructor
    public Feed() {

    }

    // constructor with parameters
    public Feed(Integer entID, String createdAT, String field1) {
        this._entID = entID;
        this._createdAT = createdAT;
        this._field1 = field1;
    }

    // All set methods

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

    public void setCreatedAT(String createdAT) {
        this._createdAT = createdAT;
    }

    public void setField1(String field1) {
        this._field1 = field1;
    }

    // All get methods

    public Integer getId() {
        return this._entID;
    }
```

```
    public String getCreatedAT() {
        return this._createdAT;
    }

    public String getField1() {
        return this._field1;
    }

    //
    @Override
    public String toString() {
        return _createdAT + " " + "\n" + _field1;
    }

}
```

APÊNDICE 5 – FEEDXMLPARSERLIST.JAVA

```
package com.example.erikakiyama.braceletapp;

import android.annotation.SuppressLint;
import android.content.Context;

import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;

import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;

public class FeedXMLParserList {

    // names of the XML tags
    private static final String CHANNEL = "channel";
    private static final String FEED = "feed";
    private static final String CREATED_AT = "created-at";
    private static final String ID = "entry-id";
    private static final String FIELD1 = "field1";

    private ArrayList<Feed> feedList = null;
    private Feed currentFeed = null;
    private boolean done = false;
    private String currentTag = null;
    private Date date;

    List<Float> Temp = new ArrayList<>();
    List<Date> Time = new ArrayList<>();

    String fullStr = null;
```

```
    public ArrayList<Feed> parse(Context context) {
        try {

            URL url = new
URL("https://api.thingspeak.com/channels/489402/feeds.xml?results=3000000")
;
            XmlPullParserFactory
factory=XmlPullParserFactory.newInstance();
            factory.setNamespaceAware(true);
            XmlPullParser parser=factory.newPullParser();

            parser.setInput(getInputStream(url), "UTF_8");

            int eventType = parser.getEventType();

            while (eventType != XmlPullParser.END_DOCUMENT && !done) {

                switch (eventType) {
                    case XmlPullParser.START_DOCUMENT:
                        feedList = new ArrayList<Feed>();
                        break;
                    case XmlPullParser.START_TAG:
                        currentTag = parser.getName();
                        if (currentTag.equalsIgnoreCase(FEED)) {
                            currentFeed = new Feed();
                        } else if (currentFeed != null) {
                            if (currentTag.equalsIgnoreCase(ID)) {

currentFeed.setId(Integer.parseInt(parser.nextText()));
                            } else if
(currentTag.equalsIgnoreCase(CREATED_AT)) {
                                DateFormat dateFormat = new
SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss");
                                date = dateFormat.parse(parser.nextText());
                                @SuppressLint("SimpleDateFormat")
DateFormat formatterGraph = new SimpleDateFormat("E, dd/M, HH:mm:ss"); //If
you need time just put specific format for time like 'HH:mm:ss'
                                Calendar calendar = Calendar.getInstance();
                                calendar.setTime(date);
                                calendar.add(Calendar.HOUR_OF_DAY, -2);
```

```
                                date = calendar.getTime();
                                Time.add(date);

currentFeed.setCreatedAT(String.valueOf(fullStr));
                                fullStr = formatterGraph.format(date);
                            } else if (currentTag.equalsIgnoreCase(FIELD1))
{
                                float temp =
Float.parseFloat(String.valueOf(parser.nextText()));

currentFeed.setField1(String.valueOf(temp));
                                Temp.add(temp);
                            }
                        }
                        break;
                    case XmlPullParser.END_TAG:
                        currentTag = parser.getName();
                        if (currentTag.equalsIgnoreCase(FEED) &&
currentFeed != null) {
                            feedList.add(currentFeed);
                        } else if (currentTag.equalsIgnoreCase(CHANNEL)) {
                            done = true;
                        }
                        break;
                }
                eventType = parser.next();
            }

        } catch (XmlPullParserException | IOException e) {
            e.printStackTrace();
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return feedList;
    }

    public InputStream getInputStream(URL url) {
        try {
            return url.openConnection().getInputStream();
        } catch (IOException e) {
            return null;
```

```
            }
        }

        public List<Float> getTemp() {
            return Temp;
        }

        public List<Date> getTime() {
            return Time;
        }

        public String getFullStr() {
            return fullStr;
        }


}
```

APÊNDICE 6 – LISTVIEWACTIVITY.JAVA

```
package com.example.erikakiyama.braceletapp;

import android.annotation.SuppressLint;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;

import java.util.Collections;
import java.util.List;

public class ListViewActivity extends AppCompatActivity {

    public List<Feed> channel = null;
    Handler h = new Handler();
    int delay = 15*1000; //1 second=1000 milisecond, 15*1000=15seconds
    Runnable runnable;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_view);

        Button GoBck = findViewById(R.id.GoBack_btn);

        GoBck.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(getApplicationContext(),
MainScreenActivity.class));
            }
        });
        new LoadFeedTask2().execute();
```

```
    }

    @Override
    protected void onResume() {
        //start handler as activity become visible
        h.postDelayed( runnable = new Runnable() {
            public void run() {
                new LoadFeedTask2().execute();
                h.postDelayed(runnable, delay);
            }
        }, delay);

        super.onResume();
    }

    @Override
    protected void onPause() {
        h.removeCallbacks(runnable); //stop handler when activity not
visible
        super.onPause();
    }


    @SuppressLint("StaticFieldLeak")
    private class LoadFeedTask2 extends AsyncTask<String, Void, List<Feed>>
{

        @Override
        protected void onPreExecute() {
            // Getting reference to the TextView tv_counter of the layout
activity_main
            TextView desc = findViewById(R.id.Description_tv);
            desc.setText("Please Wait! Updating...");
        }

        @Override
        protected List<Feed> doInBackground(String... args) {
            FeedXMLParserList parser = new FeedXMLParserList();
            channel = parser.parse(getBaseContext());
            return channel;
        }
```

```
        @Override
        protected void onPostExecute(List<Feed> employees) {
            ArrayAdapter<Feed> adapter = new
ArrayAdapter<Feed>(getBaseContext(), android.R.layout.simple_list_item_1,
employees);
            ListView listView = findViewById(R.id.TempList_tv);
            Collections.reverse(employees);
            listView.setAdapter(adapter);


            TextView desc = findViewById(R.id.Description_tv);
            desc.setText("Here are listed all the temperature values for
your device");
        }
    }
}
```

APÊNDICE 7 – MAINSCREENACTIVITY.JAVA

```
package com.example.erikakiyama.braceletapp;

import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.graphics.Color;
import android.net.ConnectivityManager;
import android.net.NetworkInfo;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.LegendRenderer;
import com.jjoe64.graphview.helper.DateAsXAxisLabelFormatter;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.DataPointInterface;
import com.jjoe64.graphview.series.LineGraphSeries;
import com.jjoe64.graphview.series.OnDataPointTapListener;
import com.jjoe64.graphview.series.Series;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.List;


public class MainScreenActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {

        if(isConnectionAvailable()){
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main_screen);
```

```
            new LoadFeedTask().execute();

            Button AddDev = findViewById(R.id.AddDevice_btn);
            Button DtList = findViewById(R.id.DataList_btn);
            Button Refresh = findViewById(R.id.Refresh_btn);

            AddDev.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    startActivity(new Intent(getApplicationContext(),
AddDeviceScreenActivity.class));
                }
            });

            DtList.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    startActivity(new Intent(getApplicationContext(),
ListViewActivity.class));
                }
            });

            Refresh.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    new LoadFeedTask().execute();
                }
            });
        }

        else {

            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_main_screen);
            Toast.makeText(this, "Not connected",
Toast.LENGTH_SHORT).show();
        }
    }

    public class LoadFeedTask extends AsyncTask<Void, Void, Void>{
```

```
        private String mStringTextView;
        private List<Float> Temp;
        private List<Date> Time;
        private String fullStr;
        private int listSize;
        final GraphView graph = findViewById(R.id.graph);

        @Override
        protected void onPreExecute() {
            TextView mTextView = findViewById(R.id.MainTemp_tv);
            mTextView.setText("Updating...");

            TextView dTextView = findViewById(R.id.Timestamp_tv);
            dTextView.setText("");
        }

        @Override
        public Void doInBackground(Void... params) {

            FeedXMLParserList parser = new FeedXMLParserList();

            List<Float> Temp = parser.getTemp();

            parser.parse(getBaseContext());
            listSize = Temp.size();

            this.fullStr = parser.getFullStr();
            this.Temp = parser.getTemp();
            this.Time = parser.getTime();
            this.listSize = Temp.size();
            this.mStringTextView = String.valueOf(Temp.get(listSize-1));
            return null;
        }

        @Override
        protected void onPostExecute(Void result){
            TextView mTextView = findViewById(R.id.MainTemp_tv);
            mTextView.setText("Last Temperature: " +
String.valueOf(this.mStringTextView) + "°C");

            TextView dTextView = findViewById(R.id.Timestamp_tv);
```

```
            dTextView.setText("Date: " + String.valueOf(this.fullStr));

            createGraph(graph, Temp, Time, listSize);
        }
    }

    protected void createGraph (final GraphView graph, List<Float> Temp,
List<Date> Time, int listSize){

        DataPoint[] dataPoints = new DataPoint[listSize];
        for (int i = 0; i < listSize; i++) {
            dataPoints[i] = new DataPoint(Time.get(i), Temp.get(i));
        }

        DataPoint[] dataPoints1 = new DataPoint[listSize];
        for (int i = 0; i < listSize; i++) {
            dataPoints1[i] = new DataPoint(Time.get(i), 37.50);
        }
        graph.removeAllSeries();
        DateFormat formatter = new SimpleDateFormat("HH:mm:ss");

        graph.getGridLabelRenderer().setLabelFormatter(new
DateAsXAxisLabelFormatter(graph.getContext(), formatter));
        // set manual X bounds
        graph.getViewport().setYAxisBoundsManual(true);
        graph.getViewport().setMinY(0);
        graph.getViewport().setMaxY(42);
        graph.getGridLabelRenderer().setNumHorizontalLabels(3); // only 3
because of the space

        Calendar calendar = Calendar.getInstance();
        Date lastDate = Time.get(listSize-1);
        calendar.setTime(lastDate);
        Date d1 = calendar.getTime();
        calendar.add(Calendar.HOUR_OF_DAY, -1);
        Date d2 = calendar.getTime();

        // set manual x bounds to have nice steps
        graph.getViewport().setMinX(d2.getTime());
        graph.getViewport().setMaxX(d1.getTime());
```

```
        graph.getViewport().setXAxisBoundsManual(true);

        // enable scaling and scrolling
        graph.getViewport().setScalable(true);

        LineGraphSeries<DataPoint> series = new
LineGraphSeries<>(dataPoints); // This one should be obvious right? :)
        LineGraphSeries<DataPoint> series2 = new
LineGraphSeries<>(dataPoints1);//
        series.setTitle("Temperature");
        series2.setTitle("Threshold Limit");
        // register tap on series callback
        series.setOnDataPointTapListener(new OnDataPointTapListener() {
            @Override
            public void onTap(Series series, DataPointInterface dataPoint)
{
                DateFormat formatterGraph = new SimpleDateFormat("EE dd/MM
HH:mm:ss"); //If you need time just put specific format for time like
'HH:mm:ss'
                String fullStr = formatterGraph.format(dataPoint.getX());
                Toast.makeText(graph.getContext(), "[" + fullStr +" ,
"+dataPoint.getY() +"]", Toast.LENGTH_SHORT).show();
            }
        });

        series2.setDrawAsPath(true);

        series.setColor(Color.BLUE);

        series2.setColor(Color.RED);
        series.setDataPointsRadius(10);
;

        graph.getLegendRenderer().setVisible(true);

graph.getLegendRenderer().setAlign(LegendRenderer.LegendAlign.BOTTOM);

        graph.addSeries(series);
        graph.addSeries(series2);
    }
```

```
    public boolean isConnectionAvailable() {
        ConnectivityManager manager = (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetworkInfo = manager.getActiveNetworkInfo();
        return (activeNetworkInfo != null) &&
activeNetworkInfo.isConnected();
    }
}
```

APÊNDICE 8 – ACTIVITY_LIST_VIEW.XML

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ListViewActivity">

    <ListView
        android:id="@+id/TempList_tv"
        android:layout_width="338dp"
        android:layout_height="483dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />

    <Button
        android:id="@+id/GoBack_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginBottom="8dp"
        android:text="Go Back"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.89"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.982" />
```

```
    <TextView
        android:id="@+id/Description_tv"
        android:layout_width="305dp"
        android:layout_height="90dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="Here are listed all the temperature values for your
device"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.041" />
</android.support.constraint.ConstraintLayout>
```

APÊNDICE 9 – ACTIVITY_MAIN_SCREEN.XML

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.design.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainScreenActivity">

    <include layout="@layout/content_main_screen" />

</android.support.design.widget.CoordinatorLayout>
```

APÊNDICE 10 – ADD_DEVICE_SCREEN.XML

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/AddChannel_tv"
        android:layout_width="wrap_content"
        android:layout_height="52dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="Here you can configure your device, Wi-Fi Settings,
Channel ID and API Key."
        android:textAlignment="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
```

```
        app:layout_constraintVertical_bias="0.073" />

    <Button
        android:id="@+id/WifiSelScrn_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="Connect to Device Wi-Fi"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.51"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.404" />

    <Button
        android:id="@+id/MainScreen_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="GoBack"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0" />

    <Button
        android:id="@+id/BrowserOpen_btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
```

```
        android:text="Set SSID and password"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.503"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.61" />

</android.support.constraint.ConstraintLayout>
```

APÊNDICE 11 – CONCENT_MAIN_SCREEN.XML

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".MainScreenActivity"
    tools:showIn="@layout/activity_main_screen">

    <TextView
        android:id="@+id/MainTemp_tv"
        android:layout_width="wrap_content"
        android:layout_height="33dp"
        android:text="Temperature"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.033" />

    <TextView
        android:id="@+id/Timestamp_tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="DateTime"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.501"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.08" />
```

```
<Button
    android:id="@+id/AddDevice_btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:text="Add Device"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.925"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.866" />

<Button
    android:id="@+id/DataList_btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
    android:text="Temperature List"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.921"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.984" />


<com.jjoe64.graphview.GraphView
    android:id="@+id/graph"
    android:layout_width="match_parent"
    android:layout_height="363dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="8dp"
    android:layout_marginBottom="8dp"
```

```
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.388" />

    <Button
        android:id="@+id/Refresh_btn"
        style="@style/Widget.AppCompat.Button"
        android:layout_width="wrap_content"
        android:layout_height="48dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="Refresh"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.183"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.981" />

</android.support.constraint.ConstraintLayout>
```