

UNIVERSIDADE DO VALE DO SAPUCAÍ
GUILHERME SANCHES PEREIRA
JÉSSICA ADRIELE DO NASCIMENTO

GERENCIADOR DE CONSUMO DE ENERGIA ELÉTRICA RESIDENCIAL

POUSO ALEGRE, MG

2016

UNIVERSIDADE DO VALE DO SAPUCAÍ
GUILHERME SANCHES PEREIRA
JÉSSICA ADRIELE DO NASCIMENTO

GERENCIADOR DE CONSUMO DE ENERGIA ELÉTRICA RESIDENCIAL

Trabalho de Conclusão de Curso apresentado ao
Curso de Sistemas de Informação da Universidade do
Vale do Sapucaí como requisito parcial para obtenção
do título de bacharel em Sistemas de Informação.

Orientador: Prof. Artur Luis Ribas Barbosa

POUSO ALEGRE, MG

2016

LISTA DE TABELAS

Tabela 1 - Custo de materiais adquiridos

42

LISTA DE QUADROS

Quadro 1 - Listagem de ferramentas	43
Quadro 2 - Listagem de ferramentas para Raspberry	44
Quadro 3 - Atualizar dependências	48
Quadro 4 - <i>Download</i> Nodejs para <i>Linux</i>	48
Quadro 5 - Instalação Nodejs	49
Quadro 6 - Verificar versão instalado	49
Quadro 7 - Instalação mongodb	49
Quadro 8 - Acessar Shell do MongoDB	50
Quadro 9 - Criação da coleção	50
Quadro 10 - Criação do documento para sensor 1	50
Quadro 11 - Instalação cliente MongoDB	50
Quadro 12 - Instalação módulo Request	50
Quadro 13 - Instalação serialport	51
Quadro 14 - Instalação ExpressJS	58
Quadro 15 - Criação do arquivo package.json	59
Quadro 16 - Instalação do AngularJS	63
Quadro 17 - Instalação angular-route	64
Quadro 18 - Instalação jQuery	65
Quadro 19 - Instalação servidor HTTP	65
Quadro 20 - Instalação Angular Material	66

LISTA DE FIGURAS

Figura 1 - <i>Raspberry Pi 2</i>	14
Figura 2 - Modelo plataforma Arduino UNO REV 3.	15
Figura 3 - Modelo de Sensor de Corrente TC SCT 013-000	17
Figura 4 - Ciclo de funcionamento interno Node.js	19
Figura 5 - Modelo gráfico highcharts - Column, line , and pie	25
Figura 6 - Tabela de valor por KWh	33
Figura 7 - Caso de uso	40
Figura 8 - Arquitetura final da solução	41
Figura 9 - SDFormatter 4	45
Figura 10 - Tela opções SDFormatter	45
Figura 11 - Diretório raiz	46
Figura 12 - Tela de seleção do Sistema Operacional	47
Figura 13 - Tela de configuração da Raspberry	47
Figura 14 - Tela inicial Raspberry	48
Figura 15 - Integração sensor TC com Arduino	53
Figura 16 - Wireframe tela de login	56
Figura 17 - <i>Wireframe</i> painel principal da aplicação	56
Figura 18 - Modelamento do banco de dados	57
Figura 19 - Resultado instalação ExpressJS	59
Figura 20 - Resultado criação arquivo package.json	60
Figura 21 - Estrutura de diretórios aplicação web	62
Figura 22 - Lista de dependências	63
Figura 23 - Estrutura de pastas da aplicação	66
Figura 24 - Menu de componentes Angular Material	67
Figura 25 - Conexão entre as placas	68
Figura 26 - Medições comparativas dos equipamentos de teste	69
Figura 27 - Conexão entre os sensores e os disjuntores	70
Figura 28 - Dados persistidos no banco online	71
Figura 29 - Consumo em 16 de setembro de 2016 às 12h e 09 minutos	71
Figura 30 - Consumo em 16 de setembro de 2016 às 12h e 10 minutos	71
Figura 31 - Ferramenta de diagnóstico	72

Figura 32 - Painel de <i>login</i>	73
Figura 33 - Gráficos de apresentação de resultados	73
Figura 34 - Testes e validações com alicate amperímetro	74
Figura 35 - Gráfico de consumo por sensor	75
Figura 36 - Gráfico de consumo por sensor	75

LISTA DE CÓDIGOS

Código 1 - Login através do modulo Request	51
Código 2 - Exemplo de código leitura serial	51
Código 3 - Configuração e leitura de corrente com Arduino	54
Código 4 - Servidor básico ExpressJS	61
Código 5 - Tag de inclusão AngularJS	64
Código 6 - Exemplo de rotas utilizando angular-route	64

LISTA DE EQUAÇÕES

Equação 1 - Valor da fatura de energia elétrica em MG	33
Equação 2 - Cálculo de corrente	34
Equação 3 - Cálculo de Energia	34
Equação 4 - Cálculo de potência	34
Equação 5 - Cálculo de energia em função da potência e tempo	35
Equação 6 - Cálculo de Energia em KWh	35
Equação 7 – Cálculo da energia em período de 1 segundo	35

LISTA DE ABREVIATURAS E SIGLAS

AC	Corrente Alternada
ANEEL	Agência Nacional de Energia Elétrica
ANSI	<i>American National Standards Institute</i>
API	<i>Application Programming Interface</i>
ARM	<i>Advanced RISC Machine</i>
CPU	<i>Central Processing Unit</i>
CRUD	<i>Create Retrieve Update Delete</i>
CSS	<i>Cascading Style Sheet</i>
ECMA	<i>European Computer Manufacturers Association</i>
EPE	Empresa de Pesquisa Energética
GB	<i>Gigabyte</i>
GPL	<i>General Public License</i>
GPL	<i>Gnu General Public License</i>
HD	<i>Hard disk</i>
HDMI	Interface Multimídia de Alta Definição
HTML	<i>HyperText Markup Language</i>
HTTP	<i>Hypertext Transfer Protocol</i>
I/O	<i>Input/Output</i>
IDE	<i>Integrated Development Environment</i>
IoT	<i>Internet of Things</i>
ISO	<i>International Organization for Standardization</i>

JSON	JavaScript <i>Object Notation</i>
LCD	<i>Liquid Crystal Display</i>
MIT	<i>Massachusetts Institute of Technology</i>
MVC	<i>Model View Controller</i>
MVR	<i>Model View Routes</i>
NoSQL	<i>Not only Structured Query Language</i>
NPM	Node <i>Package Manager</i>
NTSC	<i>National Television System(s) Committee</i>
PAL	<i>Phase Alternating Line</i>
REST	<i>Representational State Transfer</i>
SD	<i>Secure Digital</i>
SQL	<i>Structured Query Language</i>
SVG	<i>Scalable Vector Graphics</i>
TC	Transformador de Corrente
URL	<i>Uniform Resource Locator</i>
USB	<i>Universal Serial Bus</i>
VML	<i>Vector Markup Language</i>
W3C	<i>World Wide Web Consortium</i>
XML	<i>eXtensible Markup Language</i>

SUMÁRIO

1	QUADRO TEÓRICO	12
1.1	Internet das Coisas	12
1.2	Microcomputador, sensores e prototipação	13
1.2.1	<i>Raspberry Pi</i>	13
1.2.2	<i>Arduino</i>	14
1.2.3	<i>Sensor de corrente</i>	16
1.3	Linguagens de programação	17
1.3.1	<i>NodeJS</i>	17
1.3.2	<i>JavaScript</i>	20
1.3.3	<i>Linguagem C</i>	20
1.4	Frameworks e bibliotecas	21
1.4.1	<i>AngularJS</i>	22
1.4.2	<i>Materialize</i>	22
1.4.3	<i>ExpressJS</i>	23
1.4.4	<i>jQuery</i>	24
1.4.5	<i>Highcharts</i>	25
1.5	Armazenamento e gerenciamento de dados	26
1.5.1	<i>MySQL</i>	26
1.5.2	<i>MongoDB</i>	26
1.6	Métodos de integração e suas tecnologias	27
1.6.1	<i>RESTful</i>	28
1.6.2	<i>JavaScript Object Notation (JSON)</i>	29
1.7	Servidores de hospedagem	30
1.7.1	<i>OpenShift</i>	30
1.7.2	<i>GitHub Pages</i>	31
1.8	Energia e corrente elétrica no Brasil	32
1.8.1	<i>Cálculo do valor da energia elétrica</i>	32
1.8.2	<i>Corrente elétrica e potência</i>	34
2	QUADRO METODOLÓGICO	36
2.1	Tipo de pesquisa	36
2.2	Contexto de pesquisa	37

2.3	Instrumentos	38
2.4	Diagramas	39
2.4.1	<i>Casos de uso</i>	39
2.5	Procedimentos e resultados	40
2.5.1	<i>Modelagem da arquitetura</i>	40
2.5.2	<i>Aquisição dos equipamentos e componentes</i>	42
2.5.3	<i>Configuração de ambiente da aplicação web e API RESTful</i>	42
2.5.4	<i>Configuração de ambiente da Raspberry</i>	43
2.5.5	<i>Instalação do sensor de corrente com Arduíno</i>	52
2.5.6	<i>Prototipação, wireframes e mockups</i>	55
2.5.7	<i>Modelagem do banco de dados</i>	57
2.5.8	<i>Desenvolvimento da API RESTful</i>	58
2.5.9	<i>Desenvolvimento da aplicação web</i>	63
3	DISCUSSÃO DE RESULTADOS	68
	REFERÊNCIAS	77

1 QUADRO TEÓRICO

Para desenvolvimento do trabalho, foram empregadas diversas tecnologias, ferramentas e conceitos que serão apresentados neste capítulo.

1.1 Internet das Coisas

A Internet das Coisas ou IoT, do inglês, *Internet of Things*, é uma rede de objetos tecnológicos, que incorpora eletrônica, *software*, sensores e conectividade com a internet, permitindo coletar e trocar dados e informações entre si e entre outros dispositivos conectados à internet, tais como: computadores, *notebooks*, *smartphones* e *tablets* (SILVEIRA, 2016).

O termo *Internet of Things* (Internet das Coisas) foi dito pela primeira vez, em 1999, por Kevin Ashton, cofundador do Auto-ID Center do *Massachusetts Institute of Technology* (MIT). Em artigo, Ashton (2009) alegou que a ideia original do termo Internet das Coisas previa a conexão de todos os objetos físicos à internet, com capacidade de absorver informações por meio de identificação por radiofrequência e tecnologias de sensoriamento - as quais permitiriam observar, identificar e compreender o mundo independentemente das pessoas e suas limitações de tempo, atenção e precisão. Como a internet atualmente está presente em quase todos os lugares, portanto, ela se torna um meio mais eficiente para a transmissão e coleta desses dados.

A internet hoje, conta com quase três bilhões de usuários conectados segundo “*Global Internet Report*” (KENDE, 2014). Segundo previsões de Stamford (2013), em 2020 o número de dispositivos conectados e interligados será de 26 bilhões. Algumas previsões mais otimistas, segundo a CISCO, prevê em 50 bilhões de objetos conectados no mesmo período, gerando uma movimentação de mercado de US\$ 14,4 trilhões até 2022 (EVANS, 2011).

Portanto, segundo Lacerda e Lima-Marques (2015, p. 161):

As inovações que surgem no âmbito da Internet das Coisas ampliam o potencial humano em diversas áreas - tais como planejamento urbano (cidades, edifícios e trânsitos inteligentes), meio ambiente (energia, água), indústria, comércio, turismo, educação, saúde, trabalho, segurança, programas sociais, governo - com soluções capazes de promover desenvolvimento econômico, sustentabilidade e qualidade de vida.

Tais fatos trazem para sociedade atual uma demanda de pessoas, órgãos e empresas privadas capazes de desenvolver e programar recursos de internet das coisas como forma de gerar lucro financeiro e se inserir em um mercado com ampla expansão. Este conceito de uma

nova geração de conectividade, alinhado ao uso de novas tecnologias e ferramentas disponíveis no mercado, justificado pelo atual estado de crescimento tecnológico tornou-se viável e eficaz o uso de recursos de *IoT* neste trabalho.

Neste sentido, foi empregado o conceito de *IoT* para fazer com que os dispositivos interligados possam medir o consumo de energia elétrica de uma residência e exibir estes dados coletados na aplicação web para que os usuários possam ter controle de seus gastos diariamente.

1.2 Microcomputador, sensores e prototipação

O presente trabalho teve uma integração entre *software* e *hardware* resultando em uma solução composta de um microcomputador, uma plataforma de prototipação juntamente com sensores conectados a ela. Adiante são descritos cada um desses componentes.

1.2.1 Raspberry Pi

Segundo Upton e Halfacree (2013, p.26), “a *Raspberry Pi* é uma máquina completa, com considerável poder de processamento, em uma placa de circuito impresso menor do que um cartão de crédito”.

Ela foi desenvolvida, em 2006, por Eben Upton, Rob Mullins, Jack Lang e Alan Mycroft, na Universidade de Laboratório de Informática de Cambridge e na empresa *Atmel ATmega*, que desenvolve placas com processadores ARM. Eben Upton e sua equipe, em 2009, oficialmente estabeleceram a *Raspberry Pi Foundation*, fundação de caridade educacional, com base no Reino Unido, que tem como meta permitir que pessoas de todas as idades possam aprender a programar e entender como funcionam os computadores (RASPBERRY PI, 2014).

O coração do dispositivo *Raspberry Pi* é o processador multimídia *Broadcom BCM2835*, que tem a maioria dos componentes do sistema montado em um único chip e é baseado na arquitetura de conjunto de instruções ARM, desenvolvida pela *Acorn Computers*, no final dos anos 80, sendo ele capaz de operar apenas com alimentação de energia de 1A e 5V, fornecida pela sua porta micro-USB. O baixo consumo do chip é traduzido, diretamente, em pouco desperdício de energia, mesmo durante tarefas complexas de processamento. Outra diferença importante entre o *Raspberry Pi* e seu desktop, além do tamanho e do preço, é o sistema operacional Linux, que possibilita fazer download de todo o código-fonte referente ao sistema operacional e realizar quaisquer alterações necessárias (UPTON e HALFACREE, 2013).

Segundo Upton e Halfacree (2013), existem dois modelos da placa *Raspberry Pi*: Modelo A e Modelo B. A diferença entre eles é que o modelo B possui uma placa Ethernet e duas portas USB, enquanto o modelo A, contém apenas uma porta USB e nenhuma Ethernet.

Nesse projeto foi utilizado o modelo *Raspberry Pi 2 model B+*, devido a sua melhor configuração de *hardware* e aplicabilidade do projeto.

A **Figura 1** apresenta o microprocessador *Raspberry Pi 2 Model B+*:

Figura 1 - *Raspberry Pi 2*



Fonte: https://www.raspberrypi.org/wp-content/uploads/2015/01/Pi2ModB1GB_-comp.jpeg

A *Raspberry Pi* recebe os dados coletados pela Arduino, apresentada na subseção 1.2.2 por meio dos sensores eletrônicos de corrente não invasivos, que serão apresentados na subseção 1.2.3 faz a medição dos valores e envia as informações para o banco de dados *online*.

1.2.2 Arduino

Arduino é uma plataforma de desenvolvimento e prototipagem *open-source*¹ que, segundo Evans, Noble e Hochenbaum (2013, p.25), “teve início no *Interaction Design Institute* de Ivrea, na Itália, em 2005”.

Esta plataforma teve sua criação destinada a ajudar estudantes de uma universidade italiana que tinha como premissa o fato de que: “[...] o preço almejado não poderia ser mais do que um estudante gastaria se saísse para comer uma pizza” (EVANS, et. al., 2013, p.25).

Após sua criação, foram vendidas rapidamente as unidades fabricadas e a mesma começou a ser amplamente divulgada nas universidades da Itália e mais tarde pelo mundo todo.

¹ OPEN-SOURCE – Software de código aberto que qualquer um pode inspecionar, modificar e melhorar.

Esta plataforma trabalha com o conceito de entrada de dados por meio de portas eletrônicas e/ou digitais, realiza-se um processamento dos dados e gera-se uma saída, sendo este procedimento executado em *loops* predefinidos pela linguagem de programação.

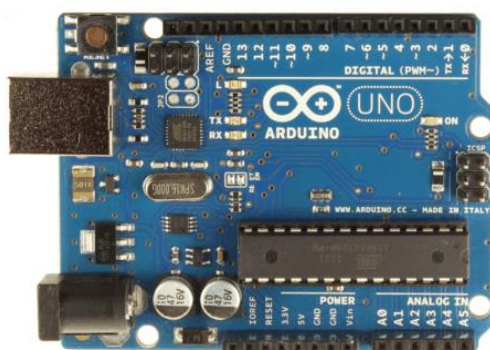
A linguagem utilizada nesta plataforma é a linguagem C, que se tornou uma linguagem adaptável a qualquer dispositivo que siga o padrão ANSI definido para a linguagem.

Segundo Arduino (2016, p.1), site oficial da plataforma, no que diz respeito à evolução de sua criação:

Ao longo dos anos, Arduino tem sido o cérebro de milhares de projetos, desde objetos do cotidiano para instrumentos científicos complexos. A comunidade mundial de tomadores - estudantes, amadores, artistas, programadores e profissionais - reuniu em torno desta plataforma *open-source*, suas contribuições acrescentaram-se a uma quantidade incrível de conhecimento acessível que pode ser de grande ajuda para novatos.

O *hardware* que compõe a plataforma Arduino é variado conforme os diversos modelos que foram lançados desde seu lançamento em 2005. Para este trabalho será analisado o modelo UNO REV 3:

Figura 2 - Modelo plataforma Arduino UNO REV 3.



Fonte: http://www.embarcados.com.br/wp-content/uploads/2013/11/imagem_01.png.

Como visto na **Figura 2**, a Arduino é composta por:

- 14 pinos de entrada e saída digital (pinos 0 - 13): que podem ser utilizados como entradas ou saídas digitais de acordo com a necessidade do projeto.
- 6 pinos de entradas analógicas (pinos A0 - A5): que são dedicados a receberem valores analógicos, por exemplo, a tensão de um sensor. O valor a ser lido deve estar na faixa de 0 a 5 V e poderá ser convertido para valores entre 0 e 1023.

- 6 pinos de saídas analógicas (pinos 3, 5, 6, 9, 10 e 11): que podem ser programados para serem utilizados como saídas analógicas.

Para desenvolver nesta plataforma, é necessário o uso de um *software* que faça *upload* do código programado em linguagem C para dentro do micro controlador embutido na placa e que o mesmo seja executado quando ela for conectada à rede elétrica. A alimentação da placa pode ser feita a partir da porta USB do micro controlador ou através de um adaptador AC com tensão recomendada de 9 a 12 volts. Segundo o *site* da equipe Embarcados (2016), após o *upload* da IDE este código feito em linguagem C é traduzido para outra linguagem que pode ser compreendida pelo micro controlador.

Neste trabalho, a *Arduino* é responsável pela conectividade com os sensores de corrente que realizam a leitura da energia elétrica e disponibilizam estes dados para a central *Raspberry* que envia os dados para a nuvem. Sua facilidade de aprendizado e compra por um baixo custo foram fatores relevantes na escolha desta plataforma.

1.2.3 Sensor de corrente

Os sensores de correntes, conhecidos também como TC (transformador de corrente), são dispositivos eletrônicos desenvolvidos para serem aplicados em diversos circuitos elétricos para mensurarem a corrente elétrica de algum dispositivo, através de plataformas de prototipagem, tais como: *Arduino* e *Raspberry*.

Ele é um componente extremamente útil, que oferece informações importantes aos microcontroladores, informando a corrente consumida pelo componente elétrico ligado, podendo assim, calcular o consumo de determinado aparelho eletrônico e também diagnosticar se determinado circuito anda consumindo mais energia do que deveria.

O princípio deste sensor é o mesmo de um transformador normal, ou seja, ele possui um ímã fixo a bobina, que gera uma tensão com o campo magnético criado pelo fio conforme a corrente é conduzida. Existem diversos modelos no mercado com diversas potências máximas de mensuração. Para o cálculo da potência consumida e obtenção dos custos com energia será utilizado um sensor de corrente não invasivo modelo SCT-013. Este sensor trabalha em uma escala de 0 a 100 ampères e temperaturas entre -25°C e 70°C segundo especificações do fabricante. Este modelo foi escolhido devido sua facilidade de ser crimpado ao circuito de testes sem a necessidade de ligação entre os fios.

Figura 3 - Modelo de Sensor de Corrente TC SCT 013-000



Fonte: <http://s3.amazonaws.com/img.iluria.com/product/18B9DA/3A3A8A/450xN.jpg>

Os sensores da família TC SCT são denominados não-invasivos, ou seja, como pode ser visto na **Figura 3**, eles possuem um encaixe lateral que se abre e fecha para a colocação do fio que precisa ser medido a corrente, este fato faz com que não haja necessidade de qualquer manuseio e modificação na fiação da residência, oferecendo maior segurança e praticidade na instalação, entretanto, devido ao isolamento dos cabos ele tem uma medida menos precisa, que pode ser resolvida com os ajustes de calibração através de cálculos até resultar em uma corrente em Amperes ou na unidade desejada.

Neste trabalho é utilizado um sensor para cada disjuntor da residência. Uma ponta do sensor de corrente é conectada ao disjuntor onde passa a corrente elétrica e a outra ponta a *protoboard*², que se conecta ao *Arduino*, responsável por receber os dados e enviar os valores de consumo de corrente de cada cômodo/disjuntor para a *Raspberry* fazer a mensuração.

1.3 Linguagens de programação

O desenvolvimento de um *software* é baseado na escrita de códigos em uma linguagem de programação específica. Dentre as mais variadas linguagens presentes e disponíveis na comunidade tecnológica foram utilizadas as linguagens descritas a seguir.

1.3.1 NodeJS

Node.js é uma plataforma construída sobre o motor *JavaScript V8*³ desenvolvido pela *Google* para facilitar o desenvolvimento de aplicações de redes rápidas e escaláveis. *Node.js*

² PROTOBOARD - É uma base plástica, contendo inúmeros orifícios destinados à inserção de terminais de componentes eletrônicos, utilizada para fazer montagens provisórias.

³ JAVASCRIPT V8 - interpretador ultrarrápido, escrito em C++ que faz download do mecanismo e o redireciona para uso no servidor.

usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ou seja, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos (MOREIRA, 2013).

Na JSConf Européia de 2009, um programador jovem chamado Ryan Dahl e mais 14 colaboradores apresentaram um projeto para solucionar um problema de bloqueio de execução em sistemas com muitas requisições simultâneas, que fazia com que um processo travava os outros até que se terminasse a execução do mesmo e assim os demais viessem a ser executados em fila. Este projeto era uma plataforma que combinava a eficiência da máquina virtual *JavaScript V8* e um laço de eventos que permitia ao servidor liberar o uso da CPU quando uma operação assíncrona⁴ fosse executada; tudo isto em especificações mínimas de *hardware*, aproveitando o poder e a simplicidade do *Javascript*, tornando aplicações assíncronas tarefas fáceis de escrever (MOREIRA, 2013).

Pereira (2015, p.2), menciona que:

Esta tecnologia possui um modelo inovador, sua arquitetura é totalmente *non-blocking thread* (não-bloqueante), apresentando uma boa performance com consumo de memória e utilizando ao máximo e de forma eficiente o poder de processamento dos servidores, principalmente em sistemas que produzem uma alta carga de processamento.

Podemos dizer que tudo em *nodejs* gira em torno do *event-loop* que controla as ações executadas na plataforma. *Node.js* possui uma fila de mensagens a serem processadas. Cada mensagem reflete o evento que a acionou e a função de *callback*⁵ que foi passada no registro desse evento. Essas mensagens são colocadas na fila de espera em decorrência de eventos, que podem ser tanto uma nova conexão HTTP como o retorno de uma operação de leitura de um arquivo feita pelo sistema operacional. Quando essas mensagens chegam na frente da fila, elas são logo processadas tornando esse processo sequencial (AGUIAR, 2015).

Quando o *Node.js*, durante a execução da função, detecta operações assíncronas, ele delega o trabalho de executar essa função para outro módulo que cria *threads*⁶ responsáveis pela execução dessas tarefas. Quando cada tarefa é completada, a função de *callback* é

⁴ ASSÍNCRONA - Termo utilizado para caracterizar a comunicação que não ocorre exatamente ao mesmo tempo, não-simultânea.

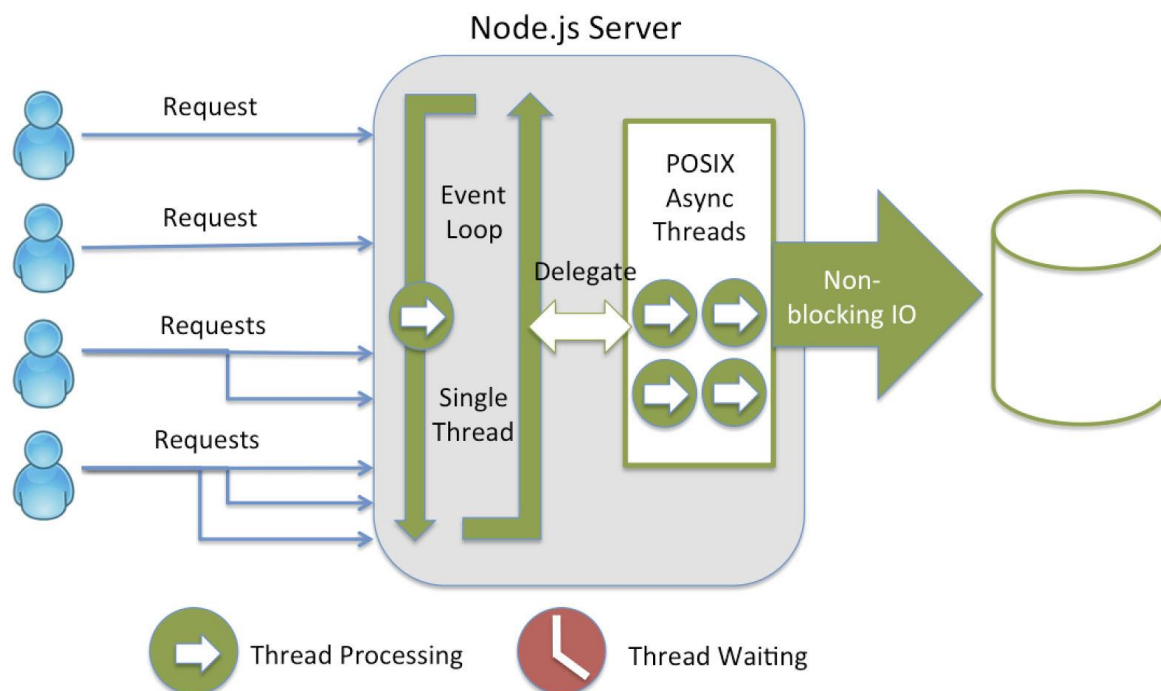
⁵ CALLBACK – É uma função passada como argumento de outra função e/ou chamada quando um evento for acontecido, ou quando uma parte de código receber uma resposta de que estava à espera.

⁶ THREAD - É um pequeno programa que trabalha como um subsistema, sendo uma forma de um processo se autodividir em duas ou mais tarefas.

transformada em uma mensagem e enfileirada na fila de mensagens, para que o *event-loop* eventualmente a execute (AGUIAR, 2015).

Na **Figura 4** é possível observar o ciclo de funcionamento interno do node.js:

Figura 4 - Ciclo de funcionamento interno Node.js



Fonte: <http://www.lazyquestion.com/images/how-node.js-works.jpg>

Sendo assim, usuários de sistemas *Node.js* estão livres de aguardarem por muito tempo o resultado de seus processos e, principalmente não sofrerão de *dead-locks*⁷ no sistema, porque nada bloqueia em sua plataforma e desenvolver sistemas nesse paradigma é simples e prático (PEREIRA, 2013).

Esta tecnologia foi aplicada no desenvolvimento do serviço *RESTful*, sendo implementada no *back-end*⁸ do projeto, pois não requer um grande volume de lógica, já que simplesmente procura valores em um banco de dados e monta uma resposta que gera um pequeno volume de tráfego utilizando somente a memória necessária para a conexão, além de ter um poder de processamento extremamente veloz, que torna eficaz a interação em tempo real entre cliente e servidor.

⁷ DEAD-LOOK – É quando duas ou mais tarefas bloqueiam uma à outra permanentemente, sendo que cada uma tem o bloqueio de um recurso que a outra tarefa está tentando bloquear.

⁸ BACK-END - Sistema responsável pela regra de negócios, webservices e API de uma aplicação.

1.3.2 JavaScript

JavaScript é uma linguagem de programação de *script*⁹ poderosa criada por Brendan Eich, contratado pela *Netscape*. Foi chamada inicialmente de *Mocha*, posteriormente de *LiveScript*, e depois nomeada de *Javascript* após a *Sun* torná-la uma marca registrada. Em pouco tempo, começou a expandir seu propósito e tornou o acesso aos elementos de HTML, como formulários, botões e imagens muito mais fácil para os desenvolvedores *Web*. A linguagem de programação começou a ser usada para manipular imagens e o conteúdo das páginas e, ao fim da década de 90, já havia *scripts* que simplesmente mudavam de uma imagem para outra em resposta a eventos gerados pela movimentação do mouse (AGUIAR, 2015).

Segundo Silva (2010), *JavaScript* tem como finalidade fornecer funcionalidades para adicionar interatividades avançadas a uma página *web*. Ela é desenvolvida para ser executada no lado do cliente, ou seja, é uma linguagem interpretada pelo navegador do usuário. A Associação Européia de Fabricantes de Computadores (ECMA) foi a responsável por criar essa especificação padrão para a linguagem, conhecida como ECMAScript. Com isto, os navegadores passaram a ter funcionalidades integradas não havendo necessidade de um compilador.

De acordo com Caelum (2015), o *JavaScript* é responsável por aplicar qualquer tipo de funcionalidade dinâmica em páginas *web*, possibilitando assim ótimos resultados. Estes resultados podem ser vistos em ferramentas como: o *gmail*, *google maps* e *google docs*.

Portanto, esta linguagem foi escolhida para ser utilizada no desenvolvimento do *front-end*¹⁰ da aplicação *web* deste trabalho que contém a interface que o cliente/usuário irá acessar para verificação de todos os dados medidos pelos sensores.

1.3.3 Linguagem C

É uma linguagem de programação que serve para escrever programas que possibilitam a comunicação entre o usuário e a máquina. Programas especiais chamados tradutores (compiladores ou interpretadores) convertem as instruções escritas na linguagem de

⁹ SCRIPT – São “roteiros” seguidos por sistemas computacionais e trazem informações que são processadas e transformadas em ações efetuadas por um programa principal.

¹⁰ FRONT-END - É responsável por coletar a entrada do usuário em várias formas e processá-la para adequá-la a uma especificação em que o back-end possa utilizar.

programação C, em instruções escritas em linguagens de máquina (0 e 1 bits), permitindo desta forma que a máquina tenha o entendimento do código (AGUILAR, 2011).

A linguagem C, foi criada em 1972 em um laboratório chamado *Bell Telephone Laboratories*, por Dennis Ritchie, para permitir a escrita de um sistema operacional, o UNIX, que utilizasse os benefícios de uma linguagem de alto nível, deixando de lado o uso de linguagens de baixo nível que exigiam maior complexidade no seu uso (DAMAS, 2007).

Inicialmente, a linguagem servia para desenvolvimento de programas de sistemas que são, segundo Schildt (1986, p.2), “[...] uma classe de programa que, ou são parte, ou operam em conjunto com o sistema operacional do computador [...]”.

Atualmente, após mais de quatro décadas de sua criação, esta linguagem se apresenta no mercado também em dispositivos eletrônicos de pequeno porte, como micro controladores, pelo fato de ser uma linguagem de fácil portabilidade. Tal expansão da linguagem C pode ser observada em Klotz (2010, p.3) que diz que “uma das melhores características da linguagem C é que não está ligada a qualquer equipamento ou sistema em particular. Isto torna mais fácil para o usuário escrever programas que serão executados sem quaisquer alterações em praticamente todas as máquinas”¹¹.

Devido a todos estes conceitos teóricos e históricos no que tange a abrangência da linguagem C em dispositivos embarcados, tornou-se viável o uso da mesma neste trabalho, atuando na plataforma de desenvolvimento embarcado, o *Arduíno*.

1.4 Frameworks e bibliotecas

Para obter o resultado esperado com maior rapidez e qualidade, optou-se por utilizar algumas bibliotecas e *frameworks*. As bibliotecas são empregadas para, na maioria das ocasiões, prover uma simplificação de criação de código nas etapas do desenvolvimento, já que as mesmas fornecem uma gama de funções e métodos pré-definidos, poupando assim, o esforço do desenvolvedor em criar todo código a partir do zero. Diferentemente das bibliotecas, em que o código deve ser adequado ao seu uso, nos *frameworks* temos a adequação do código a ele e, nesse contexto, temos então diversos recursos disponíveis ao empregá-lo.

Adiante são apresentadas as especificações das bibliotecas e *frameworks* utilizados neste trabalho.

¹¹ No original “One of the best features of C is that it is not tied to any particular hardware or system. This makes it easy for a user to write programs that will run without any changes on practically all machines”. Traduzido pelos autores.

1.4.1 AngularJS

AngularJS é um *framework open-source*, mantido pela *Google*, utilizado para estruturar o desenvolvimento *front-end* através do modelo de arquitetura *model-view-controller* (MVC). Este modelo é o padrão de arquitetura de *software* que separa a informação (e as suas regras de negócio) da interface com a qual o usuário interage.

Este *framework* é um arquivo *JavaScript* que não necessita de procedimentos de instalação e é utilizado para desenvolvimento de aplicativos *web* dinâmicos. Ele associa os elementos do documento HTML com os objetos *JavaScript*, facilitando sua manipulação e possibilitando que sua *sintaxe* seja estendida e representada de forma mais clara e sucinta (ANGULARJS, 2016).

Seu objetivo é trazer grande parte das ferramentas e capacidades que são codificadas no lado do servidor para o lado do cliente, facilitando desta forma o desenvolvimento, os testes e a manutenção dos sistemas. Ele permite também a extensão do HTML expressando as funcionalidades através de elementos personalizados, atributos, classes e comentários (FREEMAN, 2014).

Atualmente, existem duas versões do *framework*, a primeira e mais amplamente conhecida é a que foi utilizada neste projeto, enquanto a outra lançada no ano de 2016 contempla outra forma de desenvolvimento que utiliza o *TypeScript*, que nada mais é que uma abstração da linguagem *JavaScript* para facilitar o desenvolvimento e assemelhar a outras linguagens de programação como Java por exemplo, através da criação de componentes, existência de herança e variáveis com tipos definidos.

A primeira versão do *AngularJS* foi escolhida para ser utilizada neste trabalho, devido a facilidade que ela traz auxiliando no desenvolvimento *front-end* das páginas *web*.

1.4.2 Materialize

Materialize é um *framework front-end open-source* com *layout* moderno e responsivo que segue os padrões de desenvolvimento baseado em *Material Designer* da *Google*, que traz a combinação de princípios clássicos de projetos bem-sucedidos que contemplam a inovação e a tecnologia. Tem como objetivo unificar a experiência do usuário em todos os seus produtos independentemente de qual plataforma está sendo utilizada, tudo isto de forma responsiva (MATERIALIZE, 2016).

Segundo *site* oficial Materialize (2016), este framework foi criado pelos estudantes: Kevin Louie, Alan Chang, Alex Mark e Alvin Wang, da Universidade Carnegie Mellon dos Estados Unidos, para oferecer padrões de estilo escritos em CSS e *JavaScript* que pudessem ser incorporados em páginas em desenvolvimento com *design* clássico e inovador.

Materialize ajuda na construção dos componentes da *interface* para que eles fiquem com páginas *web* atraentes, consistentes e funcionais; disponibilizando ao desenvolvedor estilos e funções prontas para utilizar como e onde quiser.

Entre os recursos do *Materialize* está a criação de animações e transições, além de diferentes tipos de formulários e tabelas, barra de navegação, grande variedade de ícones; tudo isto proporcionando uma experiência rica para os usuários.

Neste trabalho, ele foi implementado de forma simples em toda aplicação de acesso aos usuários. Sua utilização ajudou na criação da interface da aplicação web de forma rápida, bonita e responsiva favorecendo a usabilidade do cliente.

1.4.3 ExpressJS

O *ExpressJS* é um dos *frameworks* mais utilizados no desenvolvimento de aplicativos *Node.js*. Ele disponibiliza um conjunto mínimo e flexível de recursos que fornecem uma robusta coleção de meios para criação das mais diversas aplicações *web* e móvel (EXPRESS, 2016).

ExpressJS é um projeto criado pela Fundação *Node.js*, que tem como missão permitir uma adoção ampla através de um modelo de governo aberto que incentiva a participação, contribuição técnica e uma estrutura de manejo a longo prazo (NODEJS, 2016).

Segundo Pereira (2013), suas principais características são:

- MVR - padrão de arquitetura de *software* que separa a aplicação nas camadas de interação do usuário (view), manipulação dos dados (model) e rotas (routes);
- MVC - padrão de arquitetura de *software* que separa a aplicação nas camadas de interação do usuário (view), manipulação dos dados (model) e controle (controller);
- Roteamento de URL via *callback*;
- *Middleware* - mediação entre outros softwares;
- *Interface* RESTful;
- Suporte a *File Uploads*;
- Integração com SQL e NoSQL;

De acordo com Brown (2014), a filosofia do *Express* é proporcionar a camada mínima entre o desenvolvedor e o servidor, para que ele tenha plena expressão de suas ideias, podendo adicionar funcionalidades ou substituí-las. Ele dá suporte a aplicações *web* de página única fazendo download do site inteiro para o cliente do navegador, ao invés de fazer requisições de rede à medida que o usuário vai navegando por páginas diferentes.

Este *framework* foi escolhido na implementação do *front-end* do trabalho, devido a facilidade de desenvolvimento que minimiza repetições de código, além das diversas funcionalidades que permitem programar conteúdo dinâmico em código *html*.

1.4.4 jQuery

jQuery é uma poderosa biblioteca *open-source* baseada em *JavaScript* capaz de adicionar interatividade e dinamismo às páginas *web* sem complicar a vida do programador, ao contrário, o objetivo proposto de acordo com o seu lema é: “Escreva menos, faça mais” (BARBOSA, 2011).

Numa tradução livre, de acordo com o próprio *site* da ferramenta, *jQuery* é rápido, pequeno e tem uma biblioteca em *JavaScript* rica em funções. Isso torna coisas como a passagem e manipulação de documentos HTML, manipulação de eventos, animação muito mais simples com uma API fácil de usar que funciona em uma enorme quantidade de navegadores (JQUEY, 2016).

Desenvolvida em 2005 por John Resig que começou a elaborar suas primeiras intenções sobre como resolver problemas de robustez de código para solução de simples problemas, seguindo os padrões web estipulados pela W3C. Em 2006, foi lançado o primeiro *plugin*¹² para a biblioteca e disponibilizado de forma pública (SILVA, 2014)

Esta biblioteca destina-se a adicionar interatividade e dinamismo às páginas *web*, proporcionando ao desenvolvedor funcionalidades necessárias à criação de *scripts* que visem a incrementar, de forma progressiva e não obstrutiva, a usabilidade, a acessibilidade e o *design*, enriquecendo a experiência de usabilidade do usuário (COSTA, 2016).

Seu uso no trabalho se justifica pela maior rapidez e usabilidade no desenvolvimento do código *front-end* destinado à aplicação *web*, proposta ao usuário final; e pela sua compatibilidade de funcionamento nos principais *browsers* (e nas suas várias versões), algo que seria complicado de gerir usando apenas o *JavaScript*.

¹² PLUGIN - Todo programa, ferramenta ou extensão que se encaixa a outro programa principal para adicionar mais funções e recursos a ele.

1.4.5 Highcharts

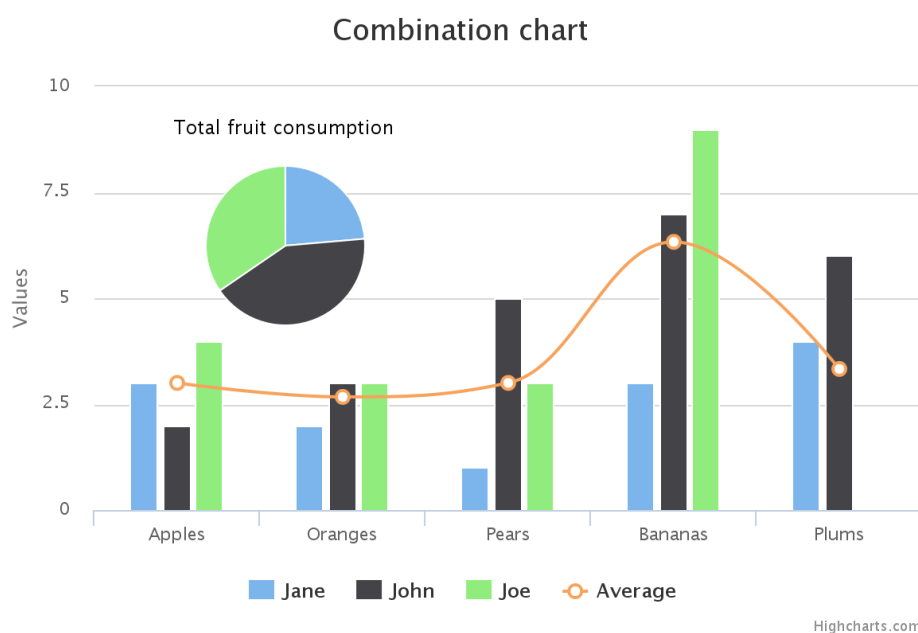
Highcharts é uma biblioteca de gráficos escrito em *JavaScript* puro, lançado em 2009, oferecendo uma maneira fácil de adicionar gráficos interativos para seu *site* ou aplicação *web*. Atualmente suporta linha, *spline* área, *areaspline*, coluna, barra, torta, dispersão, medidas de ângulo, *arearange*, *areasplinerange*, *columnrange*, bolha, gráfico de caixa, barras de erro, funil, cachoeira e tipos de gráficos polares. (HIGHCHARTS, 2016).

É um software de criação de gráficos *web* popular que produz animados gráficos impressionantes e suaves *JavaScript* e HTML5 SVG. Ele está entre o líder de *software* no mercado e tem sido utilizado em diversos setores para *sites* sociais. Embora seja construído em cima do *jQuery*, é tão simples de construir que você precisa pouca habilidade de programação para criar um gráfico *web* simples. (HIGHCHARTS, 2016).

Funciona em todos os navegadores móveis e desktop modernos, incluindo o *iPhone* / *iPad* e Internet Explorer a partir da versão 6. Nos navegadores padrões deve usar SVG para a renderização de gráficos. Em legado do Internet Explorer gráficos são desenhados usando VML. É baseado exclusivamente em tecnologias de navegador nativo e não requer *plugins* colaterais como *Flash* ou *Java*. Além disso, não precisa instalar nada no seu servidor. (HIGHCHARTS, 2016).

A **Figura 5** apresenta um modelo do gráfico *Highchart*:

Figura 5 - Modelo gráfico highcharts - Column, line , and pie



Fonte: <http://www.highcharts.com/demo/combo>.

Highcharts foi implementado no desenvolvimento deste projeto, pois tornou mais fácil a criação de gráficos interativos na geração de relatórios nas páginas da aplicação *web* e pela disponibilização dos códigos fontes gratuitamente, que nos permitiu modificações com grande flexibilidade.

1.5 Armazenamento e gerenciamento de dados

Para obter a persistência de dados dentro de um *software* é necessário armazená-los dentro de algum banco de dados, seja ele relacional, orientado a documentos, orientado a grafos ou outros. Este armazenamento é algo fundamental para qualquer solução *web* e neste trabalho foi utilizado dois tipos de banco de dados, cada um para armazenar dados específicos ao qual a informação é necessária ser persistida. Adiante temos suas especificações e referencial teórico.

1.5.1 MySQL

MySQL é o banco de dados de código aberto mais popular do mundo e possibilita a entrega econômica de aplicativos de banco de dados confiáveis, de alto desempenho e redimensionáveis, com base na *web* e incorporados (ORACLE, 2016).

Além da facilidade de uso, do alto desempenho e da confiabilidade do *mySQL*, pode-se beneficiar dos recursos avançados, das ferramentas de gerenciamento e do suporte técnico para desenvolver, implementar e gerenciar seus aplicativos.

É amplamente utilizado no projeto para armazenar os dados lidos e todas as demais configurações e informações que necessitam serem persistidas de forma *online*, como contas de usuário, históricos de acessos e outros.

1.5.2 MongoDB

MongoDB é um banco de dados orientado a documentos de código aberto que fornece alto desempenho, alta disponibilidade e escala automática. A persistência dos dados é feita através de objetos JSON com esquema dinâmico. Isso significa que você pode armazenar seus registros sem se preocupar com a estrutura de dados, com o número de campos ou tipos de campos para armazenar valores.

É um banco de dados multi-plataforma NoSQL, mantido pela empresa 10gen e foi escrito em linguagem C++. Utiliza JavaScript como interface para manipulação de dados. Pode ser executado em Windows, Linux, Mac e solares. Suporta as linguagens de programação mais populares, como: C, C++, C#, Java, PHP, JavaScript, NodeJS, Ruby e Python.

Guarda os dados em documentos ao invés de tabelas. Você pode alterar a estrutura de registros (que é chamado como documentos no MongoDB) simplesmente adicionando novos campos ou excluindo os existentes. Esta capacidade do MongoDB é útil para representar relações hierárquicas, para armazenar matrizes, e outras estruturas mais complexas de uma maneira mais simples (SOARES, 2016).

Nele trabalhamos com o conceito *schema-less*, ou seja, não existem relacionamentos de tabelas, nem chaves primárias ou estrangeiras e sim documentos que possuem documentos embutidos e tudo mantido dentro de uma coleção. Outra vantagem do *schema-less* é que os atributos são inseridos ou removidos em runtime, sem a necessidade de travar uma coleção, tornando este banco de dados flexível a grandes mudanças. Isso diminui o número de consultas complexas no banco de dados e principalmente evita criar junções para carregar diversas informações de uma vez (PEREIRA, 2013).

MongoDB introduz novos mecanismos de armazenamento que ampliam as capacidades do banco de dados, permitindo-lhe escolher as tecnologias ideais para as diferentes cargas de trabalho em sua organização. Executar vários mecanismos de armazenamento em uma implantação e alavancar a mesma linguagem de consulta, escalando métodos e ferramentas operacionais em todos eles para reduzir significativamente o desenvolvimento e complexidade (MONGODB, 2016).

A grande vantagem de trabalhar com esse modelo de banco de dados é a grande compatibilidade e suporte mantido pela comunidade própria Node.js.

1.6 Métodos de integração e suas tecnologias

Para atender aos requisitos do projeto necessitou-se o uso de uma forma de desacoplar as camadas envolvidas na solução, a fim de facilitar futuras alterações e manutenção do *software*. Estas tecnologias e conceitos são descritas abaixo.

1.6.1 RESTful

Em 2000, Roy Fielding, em sua tese de doutorado apresentou uma nova forma de integrar sistemas hipermídias distribuído chamado REST (*Representational State Transfer*) (SANTOS, 2009).

Ainda de acordo com Santos (2009), REST é um estilo de arquitetura de *software* para sistemas hipermídia distribuídos, no qual utilizamos um navegador *web* para acessar recursos, mediante a digitação de uma URL.

Requisições web possui três tipos de componentes importantes: substantivos, verbos e tipos de conteúdo. Uma aplicação *RESTful* deve dar mais importância para os substantivos do que os verbos, além de conter uma variedade de tipos de conteúdo determinados. A principal vantagem de seguir isso é que conseguimos aproveitar toda a estrutura que o protocolo HTTP proporciona (CAELUM, 2016).

a) **Substantivos**: Substantivos são os nomes dos recursos do seu sistema. Quando fazemos requisições *Web*, precisamos falar o caminho da mesma, ou seja, o identificador único de um recurso. Então ao criar substantivos do nosso sistema devemos levar em consideração que eles devem representar recursos, e não ações. Em um sistema REST, que nosso recurso chama, por exemplo, `/produtos/incluir` não estará seguindo os padrões RESTful, pois contém um verbo e não está identificando um recurso, mas sim uma operação. Portanto para representar a inclusão de um produto podemos usar apenas `/produtos` juntamente com o método HTTP *POST*, que quer dizer que estamos adicionando alguma informação no sistema referente a ao recurso produtos (CAELUM, 2016).

b) **Verbos**: Segundo Caelum (2016), dentro de um sistema *RESTful* devemos contemplar um número pequeno de operações que devem servir uma interface uniforme ao cliente/servidor. O protocolo HTTP possui sete operações: os métodos *GET*, *POST*, *PUT*, *DELETE*, *HEAD*, *OPTIONS* e *TRACE*, dentro os quais quatro deles representam o que chamamos de CRUD (*create, retrieve, update e delete*) que são:

GET - recupera informações sobre o recurso identificado. Ex: listar produtos, visualizar o produto 45. Uma requisição *GET* não deve modificar nenhum recurso você apenas recupera informações do sistema.

POST - adiciona informações. Ex: adicionar um produto. Pode adicionar informações a um recurso ou criar um novo recurso.

PUT - adiciona (ou modifica) um recurso na URI passada. Ex: atualizar um produto.

DELETE - remove o recurso representado pela URI passada. Ex: remover um produto.

c) **Tipos de Conteúdo:** Quando utilizamos uma aplicação *web* não trafegamos um recurso pela rede, simplesmente uma representação dele. Por exemplo, quando queremos adicionar um produto ao sistema, passamos para o servidor uma representação do produto: dados do formulário. E quando pedimos uma lista de produtos para o servidor ele responde com uma representação HTML desses produtos. Mas não precisa ser restrito a isso: poderíamos adicionar um produto ao sistema via uma representação em XML, e o servidor poderia nos devolver uma lista de produtos em formato JSON (CAELUM, 2016).

Resumindo: nossas chamadas devem representar recursos, as operações no recurso devem ser indicadas pelos verbos HTTP e podemos ainda falar qual é o formato (JSON, XML) em que iremos comunicar com o servidor com o *Content Type*.

Esta arquitetura e conceito foram aplicados no core da API¹³ para padronizar e desacoplar nossas camadas de desenvolvimento que são: *hardware*, serviço e aplicação *web*. Esta arquitetura nos traz manutenibilidade do sistema e benefícios para futuras mudanças na arquitetura de *hardware*.

1.6.2 JavaScript Object Notation (JSON)

JSON é um formato de dados que pode ser trocado, permutado e que se assemelha nos padrões de *scripts*, porém, não é considerado um subtipo da sintaxe do *JavaScript* (MOZILLA DEVELOPER NETWORK, 2014).

“JSON é capaz de representar números, *booleanos*, textos, listas (sequência ordenada de valores) e objetos (mapeamento de valores de texto) compostos por estes valores (ou por outras listas e objetos)” (MOZILLA DEVELOPER NETWORK, 2014, p. 1).

Um JSON está constituído em duas estruturas:

- Uma coleção de pares nome/valor.
- Uma lista ordenada de valores.

A aplicação deste formato na solução é realizada para desacoplarmos nossas entidades de *hardware*, serviço e aplicação *web*, pois este formato é entendido facilmente por seres humanos e traduzidos com a mesma facilidade nos ambientes de máquinas, ou seja, podemos trocar totalmente qualquer entidade citada anteriormente por outra, até mesmo com outros

¹³ API - Application Programming Interface (Interface de Programação de Aplicativos). É a interface, normalmente documentada que uma biblioteca ou framework disponibiliza para que o programador possa utilizá-la.

equipamentos ou outras linguagens, bastando mantermos o formato JSON de entrada e saída de dados.

1.7 Servidores de hospedagem

Para disponibilizar serviços no ambiente em nuvem, é necessário que haja uma infraestrutura de servidores robusta e boa ambientação. Para atender a necessidade do trabalho foi criado o serviço de hospedagem de *site* que será detalhado na seção a seguir.

1.7.1 OpenShift

OpenShift é uma plataforma como um serviço em nuvem que automatiza a hospedagem, a configuração, a implantação e a administração de aplicativos em um ambiente em nuvem flexível. Ela oferece aos desenvolvedores de aplicativos o acesso por autosserviço, para que eles possam implantar com facilidade os aplicativos sob demanda (REDHAT, 2016).

Aproveitando a experiência que já possui no ramo, a Red Hat lançou, em maio de 2011, durante o evento ‘Red Hat Summit’ a plataforma *OpenShift*, que cuida de toda a infraestrutura, *middleware* e gerenciamento de rede, permitindo que os desenvolvedores se concentrem apenas na criação de seus aplicativos. Este modelo de solução tecnológica suporta a implantação de *softwares* escritos em diversas linguagens como Java, Ruby, PHP, Perl, Python e Nodejs. É oferecida ainda a possibilidade de uso de uma vasta gama de servidores, bancos de dados e *frameworks*. (OPENSIFT, 2016).

Uma das principais vantagens na utilização do modelo é que ele permite ao desenvolvedor manter o foco no desenvolvimento da aplicação, realizando a implementação das aplicações de forma ágil, por meio de ferramentas disponibilizadas e possibilitando testá-las em um ambiente real de produção. É possível, ainda, mostrar, distribuir e compartilhar a aplicação funcionando sem se preocupar com a infraestrutura necessária. Tais fatores ajudam com a redução dos custos no projeto e no impacto ambiental dos recursos computacionais usados com a infraestrutura local (DIAS; JUNIOR, 2012).

Outra grande característica da plataforma é o gerenciamento do volume de conexões aos aplicativos implantados. Ela faz escalonamento automático ou manual dos recursos que apoiam as aplicações de modo que o desempenho não seja prejudicado à medida que varia o uso.

Segundo *OpenShift* (2016), existem três versões disponíveis:

- *OpenShift Online*: é a versão gratuita da plataforma, que permite aos usuários criarem até três aplicações sem custos, permitindo utilizar por aplicação até 1GB de espaço em disco e 512 MB de memória. Nesta versão o usuário pode optar por outros planos em que se paga para utilizar recursos extras.
- *OpenShift Enterprise*: é a versão empresarial, com assinatura de *software* anual, que permite ser implementada em uma nuvem privada. Ela acelera a entrega de serviços e agiliza o desenvolvimento de aplicações, oferecendo um maior grau de controle e escolha sobre os componentes podendo ser alocados mais recursos de *hardware*.
- *OpenShift Origin*: é voltado para a comunidade *open-source*, onde está disponível todo o seu código fonte para ser copiado, possibilitando ao usuário, criar a sua própria versão localmente.

Esta plataforma foi escolhida neste trabalho, pois ela possibilitou criar, implantar e gerenciar a aplicação online, fornecendo toda uma infraestrutura necessária de soluções tecnológicas.

1.7.2 GitHub Pages

GitHub é um *site* em que você pode carregar uma cópia de seu repositório git. Ele permite que você colabore facilmente com outras pessoas em um projeto. Isso é feito por meio da disponibilização de um local centralizado para compartilhar o repositório, uma *interface web* para visualizá-lo e recursos como *forking*, *pull requests* e *issues*, que permitem especificar, discutir e revisar alterações junto a sua equipe de maneira eficiente. (BELL; BEER, 2015).

Visando expandir a forma de divulgação de projetos, o *GitHub* possui um serviço chamado *GitHub Pages*, que serve principalmente para prover páginas na internet sobre os repositórios. Essas páginas podem servir para divulgar exemplos, documentações e qualquer outro tipo de informação sobre o seu projeto. E o melhor, esse serviço é de graça, basta ter uma conta no *GitHub*.

GitHub Pages é um serviço de hospedagem estática de *site* projetado para hospedar suas páginas pessoais, organizações ou projetos diretamente de um repositório *GitHub*. É possível criar e publicar *GitHub Pages online* usando o gerador de páginas automático; ou se preferir trabalhar localmente, usando o *GitHub desktop* ou a linha de comando.

O *gh-pages* é o responsável por conter os arquivos que darão vida à nossa página, assim, se o explorarmos, poderemos ver que existem arquivos como ‘index.html’, arquivos ‘.css’ entre outros. Enfim, é um branch que representa toda a página gerada.

O uso de GitHub Pages está sujeito aos Termos de Serviço do GitHub, que vem a ser uma plataforma de hospedagem de código para controle de versão e colaboração. Ele permite que você e outros trabalhem juntos em projetos de qualquer lugar. (GITHUB, 2016).

Suas capacidades incluem o seguinte:

- Documentar requisitos: Ao usar issues, é possível documentar bugs ou especificar novas funcionalidades que você queira que a sua equipe desenvolva;
- Colaborar com linhas independentes de história: Ao usar branches e pull requests, você poderá colaborar em branches ou funcionalidades diferentes.
- Revisar um trabalho em progresso: Ao observar uma lista de pull requests, você poderá ver todas as diferentes funcionalidades em que as pessoas estão trabalhando no momento e, ao clicar em qualquer pull request específico, você poderá ver as últimas alterações bem como todas as discussões em torno delas.

O GitHub é a rede para programadores. Nele podemos criar repositórios (públicos e privados) para os nossos projetos, seguir outros desenvolvedores, baixar projetos, modificar projetos, receber atualizações de modificações de projetos

Importante lembrar que o repositório GitHub é gratuito, porém é de acesso público (qualquer um pode pegar seu conteúdo), mas existe a opção de pagar pelo repositório e torná-lo privado.

Algumas empresas de tecnologia hoje procuram suas informações pelo seu GitHub ao invés de analisarem seu currículo.

1.8 Energia e corrente elétrica no Brasil

Neste tópico, detalha-se a forma que é calculado a energia elétrica, bem como a teoria sobre corrente e potência no Brasil.

1.8.1 Cálculo do valor da energia elétrica

No Brasil, os métodos de cálculo da fatura de energia elétrica se diferenciam de acordo com a concessionária de energia e a cidade em questão. Para todos os efeitos, tomou-se como base a forma de cobrança para baixa tensão praticada em Pouso Alegre – MG, que é atendida pela CEMIG (Companhia Energética de Minas Gerais). A faixa de baixa tensão se adequa para clientes residenciais, comerciais, industriais e rurais.

Para cada companhia energética, a ANEEL (Agência Nacional de Energia Elétrica) homologa constantemente a tarifa expressa em R\$/KWh (reais por quilowatt/hora) que irá vigorar para tais companhias. Essa tarifa não contempla tributos e outros elementos que fazem parte das contas de luz, tais como: ICMS, Taxa de Iluminação Pública e Encargo de Capacidade Emergencial (BRASILIA, 2014).

O quilowatt/hora é a principal variável no cálculo do valor da fatura de energia, pois, ela mede o real consumo de energia em um período de tempo. O site da CEMIG disponibiliza detalhadamente as etapas de cálculo para a fatura de energia elétrica. Dessa forma, o valor final X da fatura de energia de baixa tensão em Pouso Alegre pode ser encontrado pela **Equação 1**.

Equação 1 - Valor da fatura de energia elétrica em MG

$$X = (kWh \times THA) + CIP + EOT$$

Fonte: Elaborada pelos autores.

Onde:

X = Valor total da fatura

KWh = Consumo em quilowatts/hora





THA = Tarifa homologada pela ANEEL

CIP = Contribuição para Custeio de Iluminação Pública

EOT = Encargos e outros tributos como multas e outras cobranças que não fazem parte do cálculo básico

Atualmente, a tarifa vigente homologada pela ANEEL no ano de 2016 para a CEMIG é de R\$ 0,53122 por kWh, em bandeira verde, conforme a **Figura 6**.

Figura 6 - Tabela de valor por KWh

B1- RESIDENCIAL NORMAL	 Consumo R\$/kWh	 Consumo R\$/kWh	 PATAMAR 1 Consumo R\$/kWh	 PATAMAR 2 Consumo R\$/kWh
Residencial Normal (Consumo R\$/kWh)	0,53122	0,54622	0,56122	0,57622

Fonte: Elaborada pelos autores.

1.8.2 Corrente elétrica e potência

A corrente elétrica é por definição a taxa de fluxo de cargas elétricas através de uma superfície (TIPLER; MOSCA, 2009) – geralmente essa superfície é a seção transversal de um fio condutor. Sendo ΔQ (medida em Coulombs) a carga que flui através da área da seção transversal do fio no tempo Δt , a intensidade da corrente I medida em Ampères será:

Equação 2 - Cálculo de corrente

$$I = \frac{\Delta Q}{\Delta T}$$

Fonte: Elaborada pelos autores.

Energia é a capacidade de realizar trabalho. Ela é medida em Joules (J) ou ainda em Watt/segundo e Potência é a variação da energia (liberada ou absorvida) em função da variação do tempo, medida em watts (W) (ALEXANDER; SADIKU, 2013).

Sendo assim a Energia consumida ou fornecida será definida como visto na **Equação 3**.

Equação 3 - Calculo de Energia

$$W = P \times t$$

Fonte: Elaborada pelos autores.

Sendo:

W = Energia (watt);

P = Potência (watt por segundo ou j/s);

t = Tempo em segundos (s);

A potência será encontrada pela **Equação 4**.

Equação 4 - Cálculo de potência

$$P = V \times I$$

Fonte: Elaborada pelos autores.

Sendo:

P = Potência (watt por segundo ou j/s);

V = Tensão (Volts);

I = Corrente (A);

A princípio, o tempo utilizado nos cálculos de potência será expresso em segundos. No entanto para fins práticos, serão utilizados o Wh (watt-hora) e o kWh (quilowatt-hora). Portanto a energia consumida será dada por:

Equação 5 - Cálculo de energia em função da potência e tempo

$$Energia(Wh) = potência(W) \times tempo(h)$$

Fonte: Elaborada pelos autores.

Ou por:

Equação 6 - Cálculo de Energia em KWh

$$Energia(kWh) = \frac{potência(W) \times tempo(h)}{1000}$$

Fonte: Elaborada pelos autores.

Para se medir os dados em intervalo de 1 segundo, a fórmula passa a ser esta apresentada na **Equação 7**.

Equação 7 – Cálculo da energia em período de 1 segundo

$$Energia(kWh) = \frac{V \times I}{3600000}$$

Fonte: Elaborada pelos autores.

2 QUADRO METODOLÓGICO

Neste capítulo serão abordados e descritos os procedimentos definidos e utilizados para conduzir a pesquisa. Serão apresentados o tipo de pesquisa, seu contexto, bem como os instrumentos, sua diagramação e os procedimentos para o seu desenvolvimento.

2.1 Tipo de pesquisa

Para Tartuce (2006), método (do grego *methodos*; *met'hodos* significa, literalmente, “caminho para chegar a um fim”) é o caminho em direção a um objetivo; já a metodologia é o estudo do método, ou seja, é o corpo de regras e procedimentos estabelecidos para realizar uma pesquisa.

Gil (2007, p. 17) qualifica pesquisa como:

[...] procedimento racional e sistemático que tem como objetivo proporcionar respostas aos problemas que são propostos. A pesquisa desenvolve-se por um processo constituído de várias fases, desde a formulação do problema até a apresentação e discussão dos resultados.

De forma objetiva, a pesquisa é o meio utilizado para buscar respostas aos mais diversos tipos de indagações, ela envolve a abertura de horizontes e a apresentação de diretrizes fundamentais, que podem contribuir para o desenvolvimento do conhecimento. A pesquisa é realizada quando se tem um problema e não se tem informações suficientes para solucioná-lo.

Para o desenvolvimento deste projeto foi escolhida a pesquisa aplicada, que segundo Moresi (2003), procura gerar conhecimentos para aplicação prática, dirigidos à solução de problemas específicos, envolvendo verdades e interesses locais.

Já de acordo com Marconi e Lakatos (2009, p. 6), “Pesquisa aplicada caracteriza-se por seu interesse prático, isto é, que os resultados sejam aplicados ou utilizados, imediatamente, na solução de problemas que ocorrem na realidade”.

Tendo sido empregados estes conceitos, esta pesquisa configurou-se como aplicada, pois agregou maiores vantagens e demonstrou melhores resultados ao desenvolvimento da aplicação *web*, graças à elaboração do levantamento das informações, que possibilitaram o gerenciamento e controle do consumo de energia elétrica residencial.

2.2 Contexto de pesquisa

A energia elétrica é um insumo essencial à sociedade, indispensável ao desenvolvimento socioeconômico das nações. No Brasil, a principal fonte de geração é a hidrelétrica (água corrente dos rios), que responde por 62% da capacidade instalada em operação no país, seguida das termelétricas (gás natural, carvão mineral, combustíveis fósseis, biomassa e nuclear), com 28%. O restante é proveniente de usinas eólicas (energia dos ventos) e importação da energia de outros países. (ANEEL, 2016).

De acordo com os dados apontados pela EPE – Empresa de Pesquisa Energética, o consumo de energia elétrica no Brasil no ano de 2015 foi de 464.544.535 MWh, sendo 51.809.758 MWh do estado de Minas Gerais. Até o primeiro semestre de 2016 os dados registraram, 231.502.008 MWh, destes 12.668.620 MWh são do estado de Minas Gerais. Segundo dados do IBGE - Instituto Brasileiro de Geografia e Estatística, coletados no dia 17 de agosto de 2016, o Brasil tinha uma população de 206.307.710 habitantes, sendo 21.013.919 habitantes do estado de Minas Gerais.

Este número elevado da população fez com que o consumo de energia elétrica aumentasse e as geradoras de energia tiveram maiores gastos que foram repassados aos consumidores pelas concessionárias responsáveis. Outros fatores impactantes no custo da energia elétrica foram: o uso das usinas térmicas (mais caras), que começou em 2013, para compensar a escassez de água nos reservatórios das hidrelétricas, a falta de contratos de longo prazo - que forçou as empresas a buscarem energia no mercado livre - e assinatura de novos contratos de longo prazo já com preços mais altos.

Diante dos constantes aumentos nas tarifas de energia elétrica tornou-se de suma importância que o consumidor tivesse acesso instantâneo do valor de sua respectiva conta ao longo do mês. Munido dessas informações em tempo real, ele não levaria susto ao receber a conta no fim do mês, pois iria prever o valor conforme a evolução do consumo.

Monitorando o consumo diariamente, o consumidor também poderia iniciar um racionamento em qualquer tempo se assim achar conveniente para reduzir o valor da próxima conta, otimizando então o consumo na residência monitorada. Além dessas vantagens, problemas de medição como defeitos no medidor, equipamentos consumindo energia exageradamente e desperdícios poderiam ser detectados.

Pensando nisto, este projeto teve como objetivo o desenvolvimento de uma aplicação *web*, capaz de mostrar ao cliente final informações mais próximas dos valores reais do consumo

de energia elétrico diário e os custos acumulados ao longo do mês através de relatórios gerados em tempo real de forma prática e compreensível e facilmente acessíveis no mercado.

2.3 Instrumentos

Os instrumentos são conjuntos de ferramentas usadas para a coleta de dados. Como afirma Marconi e Lakatos (2009), eles abrangem: questionários, entrevistas e formulários.

Por questionário entende-se um conjunto de questões que são respondidas por escrito pelo pesquisado. Entrevista, por sua vez, pode ser entendida como a técnica que envolve duas pessoas numa situação "face a face" e em que uma delas formula questões e a outra responde. Formulário, por fim, pode ser definido como a técnica de coleta de dados em que o pesquisador formula questões previamente elaboradas e anota as respostas (GIL, 2007).

Outra ferramenta segundo Kioskea (2014), são as reuniões que representam um meio de compartilhamento e partilha, em um determinado grupo de pessoas, com mesmo nível de conhecimento sobre um assunto ou um problema para tomada de decisões de forma coletiva.

Reuniões entre os participantes do projeto foram contínuas, a fim de colher informações sobre o escopo do projeto, análise do progresso do desenvolvimento e validação das atividades concluídas, para se ter gerenciamento e controle dos prazos estabelecidos previamente no cronograma, evitando assim imprevistos ou situações não estabelecidas no mapa de riscos do projeto.

Com intuito de obter análise exploratória e quantitativa do número de pessoas que acreditaram na viabilização deste trabalho, aplicou-se um formulário online com questões diretamente ligadas à aprovação ou não da temática escolhida e informações relevantes sobre o público alvo do produto final.

O formulário iniciou-se com a coleta de informações a respeito da faixa etária do usuário, sua renda e gênero, seguido de questões sobre seu atual consumo de energia, bem como o seu interesse ou não em obtenção de um software para controle e gerenciamento de energia elétrica e quanto pagaria pelo mesmo. O detalhamento das questões contidas neste formulário encontra-se no capítulo de anexo.

Durante o processo de desenvolvimento desta pesquisa foram empregados instrumentos de busca de informações que trouxeram todo embasamento teórico e prático para execução do trabalho proposto. Estas ferramentas foram de enorme relevância no desenvolvimento deste trabalho apresentado, pois coletaram o máximo de informações possíveis no levantamento realizado e nortearam as tomadas de decisões.

2.4 Diagramas

A maioria dos problemas encontrados em sistemas orientados a objetos tem sua origem na construção do modelo, no desenho do sistema. Muitas vezes as empresas e profissionais não dão muita ênfase à essa fase do projeto, e acabam cometendo diversos erros de análise e modelagem. Isso quando há modelagem, pois, profissionais da área sabem que muitas vezes o projeto começa já na fase de codificação (DEVMEDIA, 2016).

2.4.1 Casos de uso

Esse diagrama documenta o que o sistema faz do ponto de vista do usuário. Em outras palavras, ele descreve as principais funcionalidades do sistema e a interação dessas funcionalidades com os usuários do mesmo sistema. Nesse diagrama não há aprofundamentos em detalhes técnicos (DEVMEDIA, 2016).

Ainda segundo Devmedia (2016) este artefato é geralmente derivado da especificação de requisitos, que por sua vez não faz parte da UML. Portanto, pode ser utilizado também para criar o documento de requisitos.

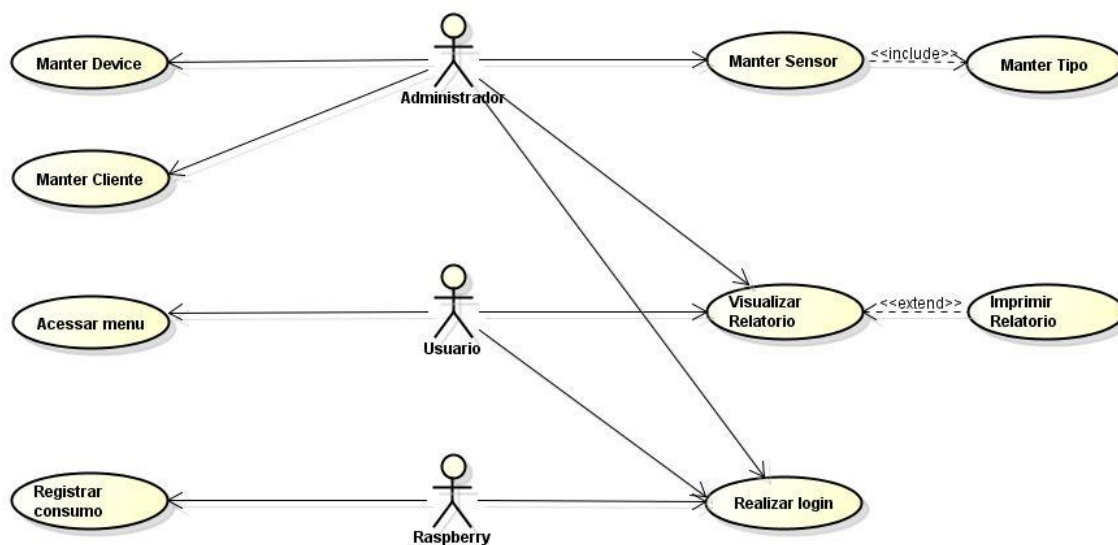
Diagramas de casos de uso são compostos basicamente por quatro partes:

- Cenário: Sequência de eventos que acontecem quando um usuário interage com o sistema.
- Ator: Usuário do sistema, ou melhor, um tipo de usuário.
- Caso de uso: É uma tarefa ou uma funcionalidade realizada pelo ator (usuário)
- Comunicação: é o que liga um ator com um caso de uso

Antes de iniciar o desenvolvimento foi efetuado a diagramação relacionando os 3 tipos de atores existentes (usuário web, Raspberry e administrador) com suas principais atividades desempenhadas no sistema.

A **Figura 7** ilustra os casos de uso do projeto juntamente com as relações com cada tipo de usuário.

Figura 7 - Caso de uso



Fonte – Elaborado pelos autores

Como pode ser visto na Figura 6, existem três atores responsáveis por atividades, algumas distintas e somente feitas por um deles, e outras comuns entre vários atores. Este fato traz para a fase de desenvolvimento uma técnica de modularizar nossa aplicação web de forma a validar níveis de acesso distintos para cada ator.

2.5 Procedimentos e resultados

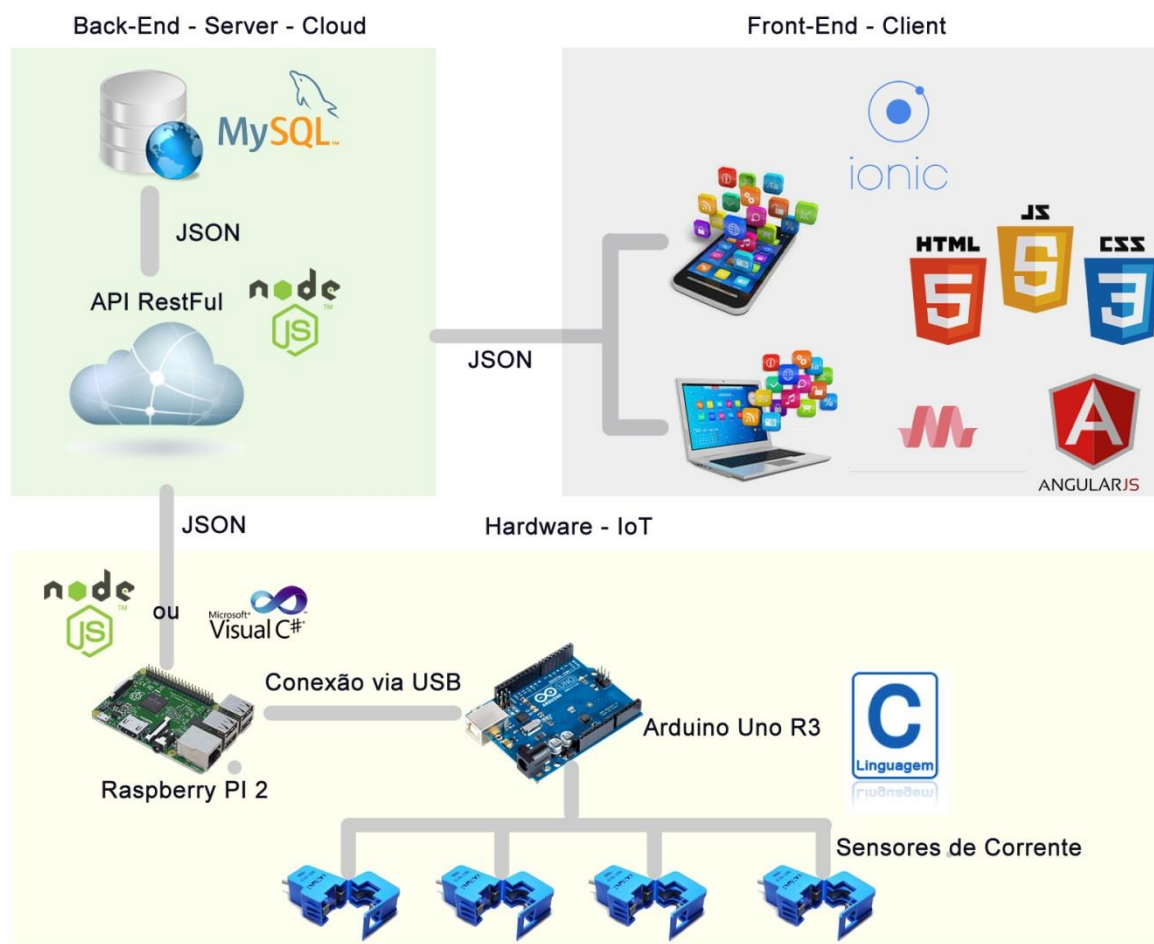
Nesta pesquisa, foi desenvolvida uma aplicação web juntamente com uma arquitetura de hardware para criar uma solução capaz de gerenciar consumo de energia elétrica de uma residência. Para organização no processo de desenvolvimento foi previamente definido um cronograma onde nele continha informações referentes ao tempo estimado para cada atividade. As atividades progrediam conforme era feito o estudo, implementação e análise dos resultados de cada tecnologia empregada no projeto.

2.5.1 Modelagem da arquitetura

O primeiro passo tomado para a criação do sistema desta pesquisa, após o levantamento das tecnologias necessárias, foi à modelagem da arquitetura que seria utilizada.

O modelo proposto pode ser observado na **Figura 8**. Como pode ser visto, os sensores de corrente não invasivo são conectados à plataforma Arduino responsável por receber os dados coletados e enviar para o microcomputador Raspberry.

Figura 8 - Arquitetura final da solução



Fonte – Elaborado pelos autores

A arquitetura desenvolvida contou com total desacoplamento entre as camadas e refletiu em resultados positivos em relação à manutenibilidade do sistema bem como sua facilidade em desenvolvimento de tarefas para a equipe, já que não havia acoplamento que gerasse conflito de atividades ou impedimentos até que um determinado ponto fosse desenvolvido.

Outro ponto referente à arquitetura, está na utilização de uma API para centralizar e disponibilizar serviços tanto para cliente final quanto para o *hardware*. Tal fator evitou replicação de códigos desnecessários e trouxe uma arquitetura mais robusta à solução final.

2.5.2 Aquisição dos equipamentos e componentes

Inicialmente foram adquiridos todos os componentes eletrônicos necessários para a concepção do modelo. Buscou-se a compra dos materiais de menor custo em várias lojas especializadas da internet e de Santa Rita do Sapucaí visando à viabilidade econômica da pesquisa. Dessa forma, obteve-se um orçamento relativamente baixo para se desenvolver um protótipo funcional (ver **Tabela 1**).

Tabela 1 - Custo de materiais adquiridos

ITEM	QTDD	PREÇO UNI	PREÇO TOTAL
SENSOR DE CORRENTE 100A	2	R\$ 20,00	R\$ 40,00
ARDUINO	1	R\$ 32,00	R\$ 32,00
FIOS JUMPER	80	R\$ 0,10	R\$ 8,00
RESISTORES	10	R\$ 0,10	R\$ 1,00
CAPACITORES	2	R\$ 1,00	R\$ 2,00
ALICATE AMPERÍMETRO	1	R\$ 159,00	R\$ 159,00
RASPBERRY PI MODEL B	1	R\$ 100,00	R\$ 100,00

Fonte – Elaborado pelos autores

Alguns equipamentos necessários ao projeto não foram listados devido à não necessidade de compra dos mesmos, outros foram doados por colegas e amigos.

2.5.3 Configuração de ambiente da aplicação web e API RESTful

Para desenvolvimento desta pesquisa foi necessário configurar um ambiente de desenvolvimento local para que fosse possível elaborar, criar protótipos, desenvolver e testar o produto final. O sistema operacional dos computadores para desenvolvimento é Windows nas versões 8.1 e 10, porém não representa nenhuma restrição caso fosse alguma outra versão do sistema operacional *Windows*.

Alguns pré-requisitos foram necessários para que houvesse sucesso no início no desenvolvimento da solução: A primeira ação tomada foi instalação das seguintes ferramentas exibidas no **Quadro 1**.

Quadro 1 - Listagem de ferramentas

Ferramenta	Versão	Disponível em	Característica
NodeJS	4.5.0	https://nodejs.org/en/download/	Linguagem de servidor
NPM	2.15.8	https://nodejs.org/en/download/	Gerenciador de dependências do NodeJS
GIT for Windows	2.9.3	https://git-scm.com/download/win	Controlador de versão
VSCode	1.4	https://code.visualstudio.com/download	Editor de texto
Arduino IDE	1.6.11	https://www.arduino.cc/en/Main/Software	IDE para upload do código para a Arduíno

Fonte – Elaborado pelos autores

Todas estas ferramentas podem ser encontradas pelos links de *download*, e após seu *download*, a sua instalação segue os padrões normais, sem nenhuma alteração, ou seja, basta clicar duas vezes no instalador e seguir os passos padrões sugeridos pelo programa.

Ao término das instalações das ferramentas o ambiente estava pronto para iniciar o desenvolvimento. Não se obteve qualquer erro nas etapas de instalação do ambiente.

2.5.4 Configuração de ambiente da Raspberry

Além da plataforma de desenvolvimento Arduíno, a pesquisa traz a integração com o microcomputador *Raspberry Pi* para coleta, armazenamento temporário e envio dos dados coletados pela Arduíno para a API RESTful onde é tratado e persistido para consulta dos clientes através da interface *web*.

Para que fosse possível iniciar qualquer desenvolvimento neste microcomputador, foi necessário primeiramente realizar a instalação de seu sistema operacional e para este procedimento necessitou-se dos seguintes itens:

- *Raspberry Pi*
- Fonte de alimentação - 5V 650ma

- Cartão SD (mínimo 4GB)
- TV com entrada HDMI ou vídeo composto
- Teclado
- Mouse
- Computador (notebook ou Desktop) (para preparar o cartão SD)

Além dos itens acima listados, necessitou-se dos seguintes softwares listados no **Quadro 2**.

Quadro 2 - Listagem de ferramentas para Raspberry

Ferramenta	Versão	Disponível em	Característica
SD Formatter	4.0.0	https://www.sdcard.org/downloads/formatter_4/	Formatação de cartões SD
Noobs	1.9.2	http://downloads.raspberrypi.org/noobs	ISO's de sistemas operacionais para Raspberry

Fonte – Elaborado pelos autores

O Raspbian é uma versão de Debian para *Raspberry* que é a mais difundida e a recomendada para todos que estão começando a utiliza-lo. Entretanto há diversos outros sistemas operacionais compatíveis com este microcomputador (Distribuições do Debian, RaspbianOS, Pidora e até mesmo Windows 10 para *IoT*).

Para realizar sua instalação na *Raspberry*, seguiu-se as seguintes etapas.

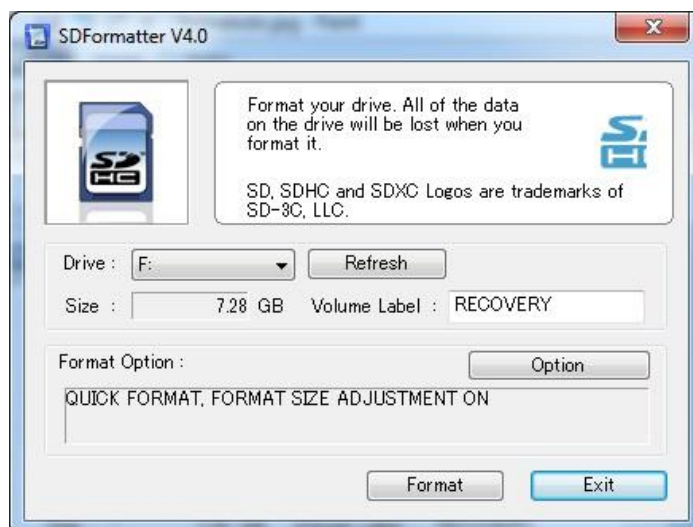
1º - Download dos arquivos e verificação da integridade dos mesmos.

Baixou-se todas as ferramentas citadas acima.

2º - Preparação do cartão SD.

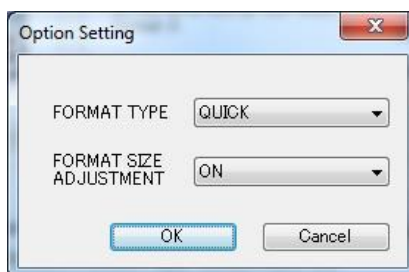
Colocou-se o cartão SD (4GB ou +) no computador e formatou-o utilizando a ferramenta SDFormatter4 (Windows ou MAC). Este procedimento é necessário para que o cartão SD seja capaz de aceitar a ISO do sistema operacional e sirva para a *Raspberry* assim como um HD (*hard disk*) serve para um computador pessoal ou *notebook*.

Na **Figura 9** mostra a tela do *software* responsável por este procedimento.

Figura 9 - SDFormatter 4

Fonte – <http://labdegaragem.com/profiles/blogs/tutorial-raspberry-pi-instalando-o-raspbian-raspberry-pi-debian>

Instalou-se o programa, abriu-o e foi selecionado a unidade onde estava o cartão SD. Após, clicado em 'Option' e alterado para 'ON' a opção '*FORMAT SIZE ADJUSTMENT*' conforme mostra a **Figura 10**.

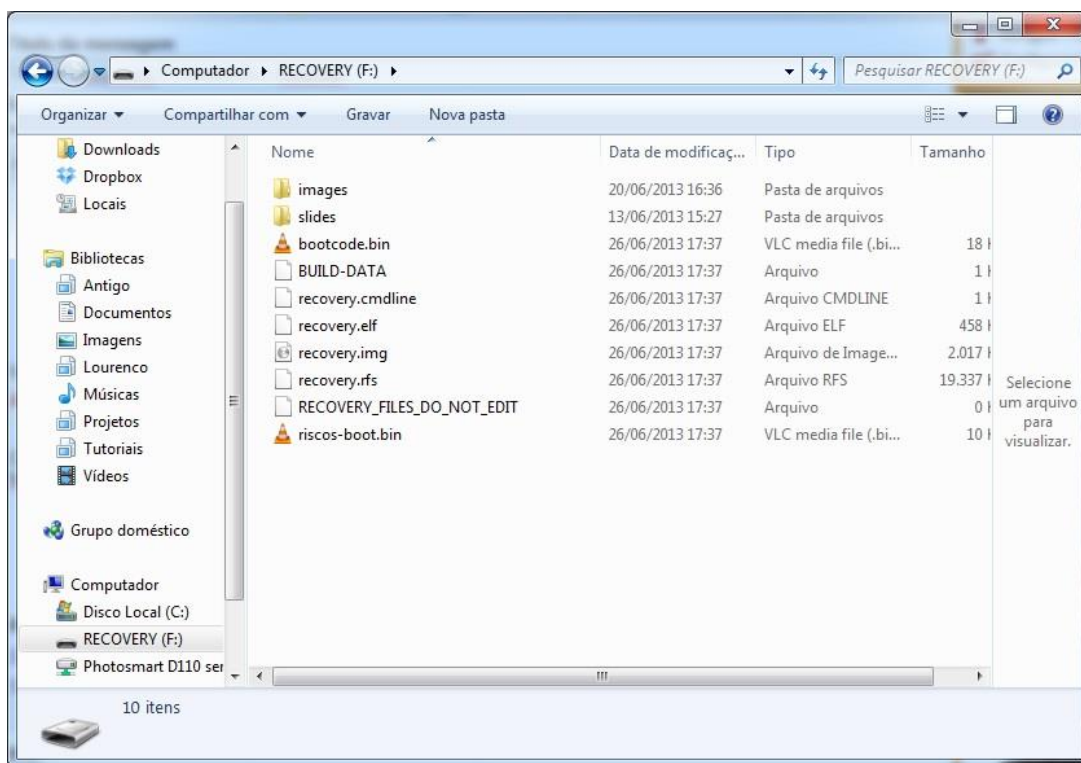
Figura 10 - Tela opções SDFormatter

Fonte – <http://labdegaragem.com/profiles/blogs/tutorial-raspberry-pi-instalando-o-raspbian-raspberry-pi-debian>

“ ATENÇÃO: Selecionando a unidade errada pode-se apagar os arquivos que não queira. É necessário cuidado ao executar esta ação ”.

Com o cartão formatado, foi utilizado um descompactador de arquivos para extrair na raiz do cartão SD, o arquivo NOOBS_v1_9_2.zip.

A raiz do cartão ficou seguindo a estrutura exibida na **Figura 11**:

Figura 11 - Diretório raiz

Fonte – <http://labdegaragem.com/profiles/blogs/tutorial-raspberry-pi-instalando-o-raspbian-raspberry-pi-debian>

3º - Ligando a *Raspberry* Pi

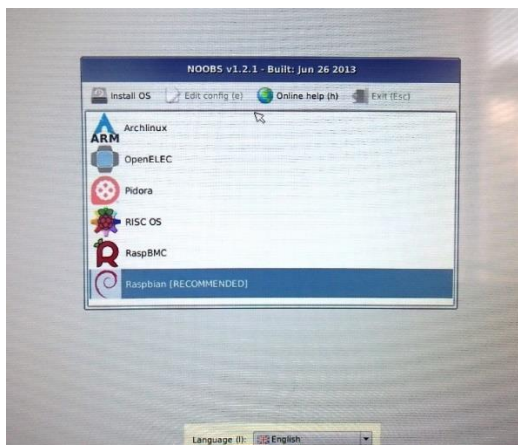
Foi conectado o mouse, teclado, cabo HDMI no monitor e o cartão SD na *Raspberry* e só depois ligado a fonte de alimentação na mesma.

Com ela ligada, precisou utilizar as teclas 1, 2, 3 e 4 do teclado, para alternar entre diferentes modos de vídeo.

Abaixo segue lista com o que cada tecla faz:

- 1- Modo Padrão HDMI
- 2- HDMI *Safe Mode* - Use este modo se não ver nenhuma imagem no modo 1
- 3- Modo vídeo composto PAL (saída de vídeo composto)
- 4- Modo vídeo composto NTSC (saída de vídeo composto)

Após as etapas acima, a **Figura 12** exibe a tela que foi exibida.

Figura 12 - Tela de seleção do Sistema Operacional

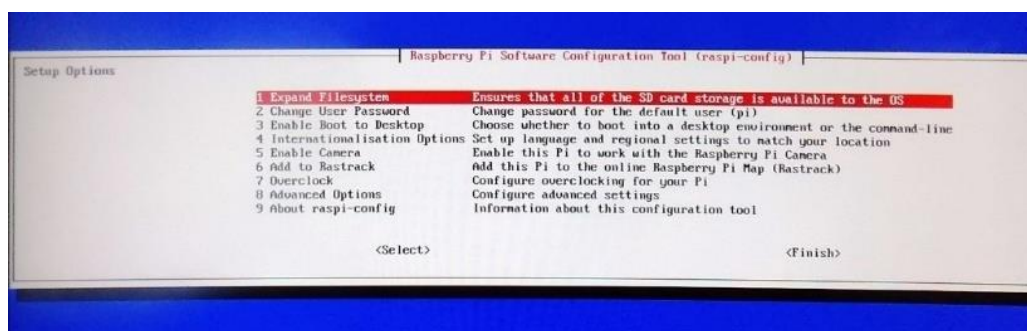
Fonte – <http://labdegaragem.com/profiles/blogs/tutorial-raspberry-pi-instalando-o-raspbian-raspberry-pi-debian>

Nesta tela listam vários sistemas operacionais disponíveis para *Raspberry Pi*, neste projeto, foi instalado o Raspbian que está marcado como recomendado.

Clicou-se nele e confirmamos a instalação. Nesta hora, foi particionado e copiado a imagem do Linux para o cartão SD.

Cerca de 10 minutos depois, uma tela confirmando que a imagem foi transferida com sucesso foi exibida.

Foi reiniciado e então exibiu-se um menu em um fundo azul, como pode ser visto na **Figura 13**.

Figura 13 - Tela de configuração da Raspberry

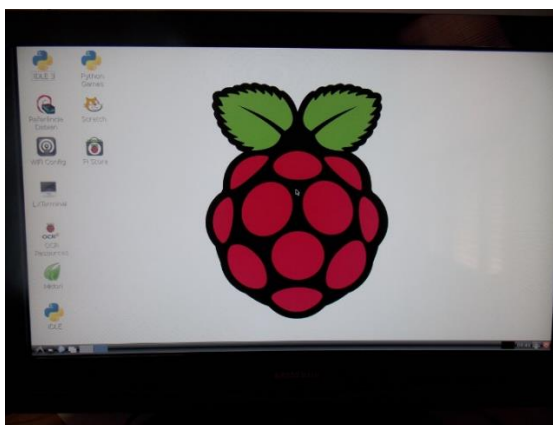
Fonte – <http://labdegaragem.com/profiles/blogs/tutorial-raspberry-pi-instalando-o-raspbian-raspberry-pi-debian>

Este é um auxiliar de configuração de sistema e nele existem várias opções para personalizar a instalação. Nesta tela pode-se configurar as opções de linguagem, a matriz do teclado, usuário e senha e diversas outras funções referentes ao sistema. Pode-se navegar utilizando as teclas: Direção, Enter, Esc e *Backspace* do teclado.

Ao finalizar as configurações, confirmamos em *Finish* (seta para esquerda para chegar até esta opção).

Novamente foi reiniciado e então a tela com a interface gráfica do Raspbian foi exibida, como pode ser visto na **Figura 14**.

Figura 14 - Tela inicial Raspberry



Fonte – <http://labdegaragem.com/profiles/blogs/tutorial-raspberry-pi-instalando-o-raspbian-raspberry-pi-debian>

Após a instalação do sistema operacional foi necessário instalar também o NodeJS no microcomputador para desenvolvimento de um serviço que está na *Raspberry* realizando o papel de central de recepção e distribuição de dados. Para este procedimento necessitou-se executar os seguintes comandos.

Quadro 3 - Atualizar dependências

```
pi@raspberrypi: ~ $ sudo apt-get update
```

Fonte – Elaborado pelos autores

O Comando do **Quadro 3** atualiza todos os pacotes do sistema operacional. Após término precisou-se recuperar a versão do NodeJS mais atualizada do repositório remoto.

Quadro 4 - Download Nodejs para Linux

```
pi@raspberrypi: ~ $ curl -sL https://deb.nodesource.com/setup_4.x | sudo  
-E bash -
```

Fonte – Elaborado pelos autores

Após término da atualização do repositório de versões do NodeJS precisou-se executar o comando mostrado no **Quadro 4**.

Quadro 5 - Instalação Nodejs

```
pi@raspberrypi: ~ $ sudo apt-get install -y nodejs
```

Fonte – Elaborado pelos autores

O comando do **Quadro 5** executa a instalação do NodeJS na Raspberry.

Para saber se tudo ocorreu como esperado foi executado o comando do Quadro 6 para verificar se realmente foi instalado com sucesso.

Quadro 6 - Verificar versão instalado

```
pi@raspberrypi: ~ $ node -v
```

Fonte – Elaborado pelos autores

O próximo passo foi a instalação do gerenciador de banco de dados não relacional MongoDB localmente na *Raspberry*. Para este procedimento, foi executado o seguinte código exibido no **Quadro 7**.

Quadro 7 - Instalação mongodb

```
pi@raspberrypi: ~ $ sudo apt-get install -y mongodb
```

Fonte – Elaborado pelos autores

Instalado o NodeJS, e o MongoDB, os scripts de execução agendada poderiam ser escritos e os dados persistidos localmente no banco de dado não relacional para caso haja alguma falha na internet, não haja perda dos dados neste período. Para persistência de dados localmente precisou-se criar uma coleção no MongoDB para salvar nossos históricos de leitura. Para este procedimento ser concluído, foi executado a seguinte linha de comando exibida no **Quadro 8** para acessa o shell do MongoDB e então criar nossa coleção chamada “Raspberry” com o comando exibido no **Quadro 9** e depois executar o comando do **Quadro 10** para criar nosso documento “*History*” para salvar os dados mensurados pela Arduino.

Quadro 8 - Acessar Shell do MongoDB

```
pi@raspberrypi: ~ $ mongo
```

Fonte – Elaborado pelos autores

Quadro 9 - Criação da coleção

```
pi@raspberrypi: ~ $ use Raspberry
```

Fonte – Elaborado pelos autores

Quadro 10 - Criação do documento para sensor 1

```
pi@raspberrypi: ~ $ db.history.save({"sensor_id": 1, "data": 0.00})
```

Fonte – Elaborado pelos autores

Os procedimentos acima obtiveram êxito e então os códigos em JavaScript foram escritos para ler os dados da serial, salvar localmente no banco de dados não relacional e então serem enviados através de *POST* para a API *online*.

Para acessar através de JavaScript os dados lidos e persistidos no banco de dados, foi necessário a instalação de um modulo cliente do MongoDB pelo comando listado no **Quadro 11**.

Quadro 11 - Instalação cliente MongoDB

```
pi@raspberrypi: ~ $ npm install --save mongodb
```

Fonte – Elaborado pelos autores

Posteriormente para poder realizar requisições HTTP para a API RESTful externa necessitou-se instalar outro modulo chamado Request. O **Quadro 12** exhibe o comando necessário para tal instalação.

Quadro 12 - Instalação módulo Request

```
pi@raspberrypi: ~ $ npm install --save request
```

Fonte – Elaborado pelos autores

Com este módulo instalado, foi possível realizar login na API *online* e receber um *token* de acesso para o cliente *Raspberry*, o **Código 1** exhibe este procedimento sendo executado.

Código 1 - Login através do modulo Request

```

1. var login = function () {
2.   request.post({
3.     uri: 'http://api-energymonitor.rhcloud.com/api/v1/login',
4.     form: { username: 'admin', pass: '*****' },
5.     method: 'POST'
6.   }, function (error, response, body) {
7.     if (!error && response.statusCode == 200) {
8.       process.env['TOKEN'] = JSON.parse(response.body).result.token;
9.       job();
10.    }
11.  });
12. };

```

Fonte – Elaborado pelos autores

Outro módulo necessário foi o *serialport*, responsável por acessar a porta serial da *Raspberry* e ler os dados escritos pela Arduino na mesma através do cabo USB. O **Quadro 13** exhibe o comando para instalação do mesmo e o **Código 2**, exemplifica seu uso.

Quadro 13 - Instalação serialport

```
pi@raspberrypi: ~ $ npm install -save serialport
```

Fonte – Elaborado pelos autores

Código 2 - Exemplo de código leitura serial

```

1. var SerialPort = require("serialport");
2. var port = new SerialPort("/dev/ttyUSB0", {
3.   baudRate: 9600
4. });
5.
6. port.on('data', function (data) {
7.   console.log('Data: ' + data);
8. });

```

Fonte – Elaborado pelos autores

A linha 1 recebe o modulo serialport e a linha 2 instancia um objeto na porta USB0 da *Raspberry* com velocidade de leitura de 9600. A linha 6 será executada toda vez que a Arduino escrever algum dado na saída serial da *Raspberry* e consequentemente neste código a linha 7 irá ser executada imprimindo no console do JavaScript os dados lidos.

2.5.5 Instalação do sensor de corrente com Arduíno

O projeto integra *hardware* com *software*, e para este procedimento obter o resultado esperado foi necessário realizar ampla pesquisa sobre os equipamentos utilizados e suas limitações com relação a integração.

A ligação do sensor TC com a plataforma Arduíno seguiu os padrões de ligação entre os componentes e esta ligação deve seguir alguns critérios e cuidados,

Segundo informações do *datasheet*, o sensor de corrente SCT-013-020 (20A) tem na saída uma variação de tensão, e o SCT-013-000 (100A e utilizado no projeto), tem na saída uma variação de corrente.

Assim, no micro controlador Arduíno conseguimos ler quase que diretamente a variação de tensão, porém no de 100A precisamos de um componente adicional: o “*burden resistor*” (“resistor de carga”), para gerar a variação de tensão que precisamos para efetuar a leitura na Arduíno.

Para calcular o resistor de carga, vamos seguir alguns passos, segundo Thomsen (2015):

1 – Determinar a corrente máxima que vamos medir

No nosso caso, é um sensor de 100A, logo vamos determinar esse valor como corrente máxima

2 – Converter a corrente máxima RMS para corrente de pico, multiplicando-a por $\sqrt{2}$

$$\text{corrente-pico-primaria} = \text{corrente-RMS} \times \sqrt{2} = 100 \text{ A} \times 1.414 = 141.4 \text{ A}$$

3 – Dividir a corrente de pico pelo número de voltas do CT (2000) para determinar a corrente de pico na bobina secundária:

$$\text{corrente-pico-secundaria} = \text{corrente-pico-primaria} / \text{numero-voltas} = 141.4 \text{ A} / 2000 = 0.0707 \text{ A}$$

4 – Para melhorar a resolução da medição, a voltagem através do resistor de carga no pico de corrente deve ser igual a metade da tensão de referência do Arduíno (AREF/2). Como a tensão de referência no Arduíno é de 5V, teremos:

$$\begin{aligned} \text{resistor-carga-ideal} &= (\text{AREF}/2) / \text{corrente-pico-secundaria} = 2.5 \text{ V} / 0.0707 \text{ A} \\ &= 35.4 \Omega \end{aligned}$$

Resumindo o cálculo anterior:

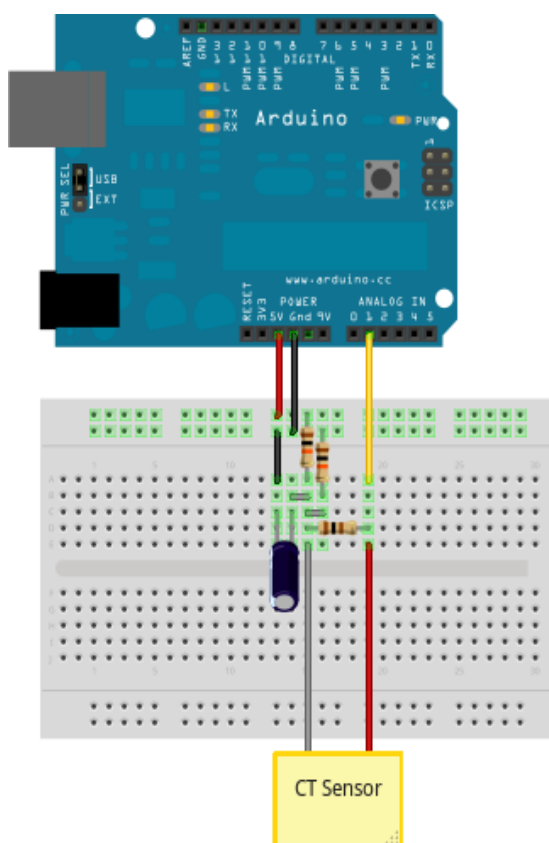
Resistor de carga (ohms) = $(AREF * CT-VOLTAS) / (2\sqrt{2} * \text{corrente-máxima-pico})$, onde CT-VOLTAS refere-se ao número de voltas da bobina interna do sensor (este dado se encontra no *datasheet* do mesmo).

Após obter os dados necessários para montagem do circuito, foi realizada a prototipação através de uma *protoboard*. Os componentes necessários, foram:

- Sensor de Corrente SCT-013-000
- 2 Resistores de 10K
- 1 Resistor de 33 Ω (para o resistor de carga)
- 1 Capacitor 10 μF

A **Figura 15** ilustra o circuito construído:

Figura 15 - Integração sensor TC com Arduino



Fonte – <http://blog.filipeflop.com/arduino/medidor-de-energia-eletrica-com-arduino.html>

O próximo passo foi a abertura do programa ArduinoIDE já instalado na preparação do ambiente e a criação do seguinte código exibido no **Código 3**.

Código 3 - Configuração e leitura de corrente com Arduino

```

1. //Carrega as bibliotecas
2. #include "EmonLib.h"
3.
4. //instancia biblioteca para leitura da corrente
5. EnergyMonitor emon1;
6. EnergyMonitor emon2;
7.
8. //variveis globais
9. int rede = 125.4;
10. int pino_tomadas = 0;
11. int pino_lampadas = 1;
12. unsigned long ltmillis, tmillis, timems, timeaux;
13.
14. void setup() {
15.     Serial.begin(9600);
16.     emon1.current(pino_tomadas, 60.2); //Pino, calibracao
17.     emon2.current(pino_lampadas, 60.2); //Pino, calibracao
18. }
19.
20. void loop() {
21.     // Calcula quantidade de tempo desde a última mensuração.
22.     ltmillis = tmillis;
23.     tmillis = millis();
24.     timems = tmillis - ltmillis;
25.
26.     double Irms = emon1.calcIrms(1480); // Cálculo da corrente do sensor_1
27.     double Irms2 = emon2.calcIrms(1480); // Cálculo da corrente do sensor_2
28.     double kwhTemp2 = 0.0, kwhTemp = 0.0; //variaveis temporarias
29.
30. //caso nao exista nenhum fio para medir, o sensor le um valor inválido entre 0 a 0.3A
31.     if (Irms > 0.3 || Irms2 > 0.3) {
32.
33.         if (Irms > 0.3) {
34.             //conversão do valor lido em amperes para quilowatts/hora do primeiro sensor
35.             kwhTemp = (((Irms*127.0)/1000.0) * 1.0/3600.0 * (timems/1000.0));
36.
37.             if (Irms2 > 0.3) {
38.                 //conversão do valor lido em amperes para quilowatts/hora do segundo sensor
39.                 kwhTemp2 = (((Irms2*127.0)/1000.0) * 1.0/3600.0 * (timems/1000.0));
40.
41.                 //impressão em formato sensor_1:valor_1,sensor_2:valor_2
42.                 Serial.print("0:");
43.                 Serial.print(kwhTemp, 10);
44.                 Serial.print(",1:");
45.                 Serial.print(kwhTemp2, 10);
46.                 Serial.println("");
47.
48.             } else {
49.                 //impressão em formato sensor_1:valor_1,sensor_2:valor_2
50.                 Serial.print("0:");
51.                 Serial.print(kwhTemp, 10);
52.                 Serial.print(",1:");
53.                 Serial.print(kwhTemp2, 10);
54.                 Serial.println("");
55.             }
56.             delay(1000);
57. }

```

Fonte – Elaborado pelos autores

O **Código 3** foi comentado para melhor entendimento por outros usuários. Neste exemplo a Arduíno realiza a cada 1 segundo a leitura de corrente do fio inserido por dentro do sensor de corrente. O resultado da leitura é exibido no Serial Monitor da própria IDE da Arduíno. Ao realizar a integração com a Raspberry, a mesma recebe estes dados via conexão USB. Para realizar o *upload* deste código para a Arduíno bastou selecionar a opção de *UPLOAD* disponível na IDE.

2.5.6 Prototipação, wireframes e mockups

Boas práticas de programação estão presentes ao redor dos desenvolvedores como uma forma de otimizar o desenvolvimento e facilitar o entendimento de qualquer desenvolvedor ao ler um código de outro devido ao fato de considerar padrões de programação, arquitetura e desenvolvimento seguindo práticas que são consideráveis as melhores para desenvolvimento *web*.

Uma dessas boas práticas é a construção e elaboração de imagens prévias que representam como o sistema final deve estar. Estas imagens são chamadas de *wireframes* ou *mockups*; caso elas representem um *designer* de baixa fidelidade do produto final é considerado *wireframes*; caso representem o produto final com média e alta fidelidade, são consideradas *mockups*.

Nesta pesquisa foi elaborado alguns *wireframes* para prototipação de uma aplicação de testes para se analisar os impedimentos e facilidades que poderiam haver durante todo o desenvolvimento.

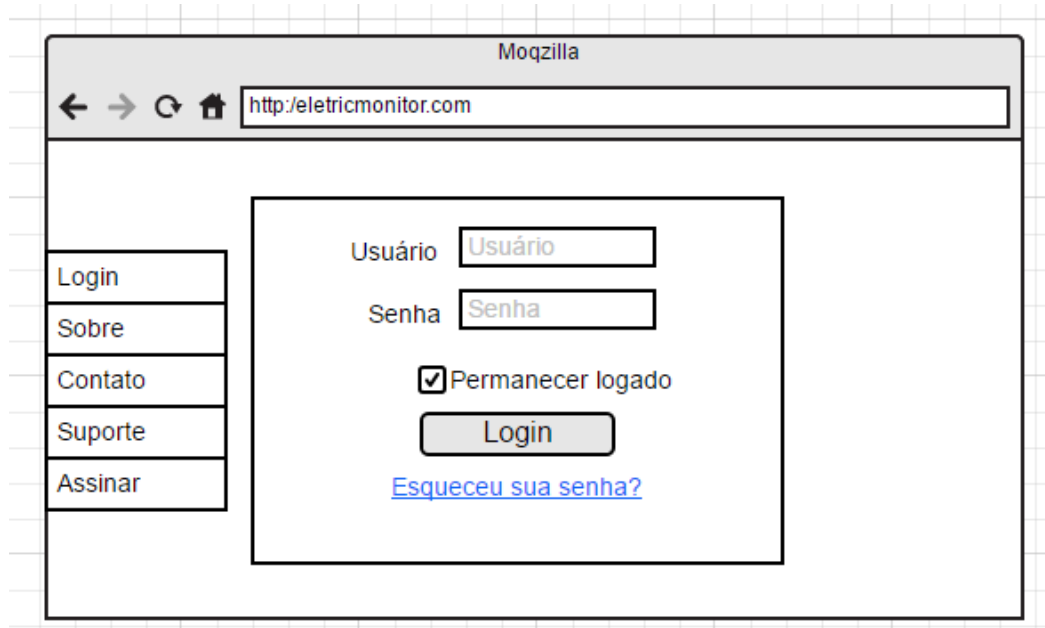
Ao término da prototipação do sistema, foi colhido alguns *feedbacks* positivos como rapidez no desenvolvimento seguindo o *wireframes* e outros negativos como dificuldade em posicionar alguns componentes presentes no *wireframes* devido ao utilizarmos o padrão de *Material Designer* como *framework* de *layout web*.

Abaixo segue 2 *wireframes* criados para criação da aplicação *web*.

Os mesmos foram gerados através de uma aplicação *online* com versão paga e gratuita com recursos restritos (<http://mockuos.com>).

O primeiro *wireframes* refere-se à **Figura 16** ela exibe a tela de login com algumas funcionalidades básicas de acesso necessárias aos usuários. Além destas funcionalidades, percebe-se o layout do menu e da caixa de login de forma centralizada ao centro e no menu centralizado ao centro verticalmente. Estas previsões de posicionamento foram fundamentais na construção e desenvolvimento da tela final da aplicação.

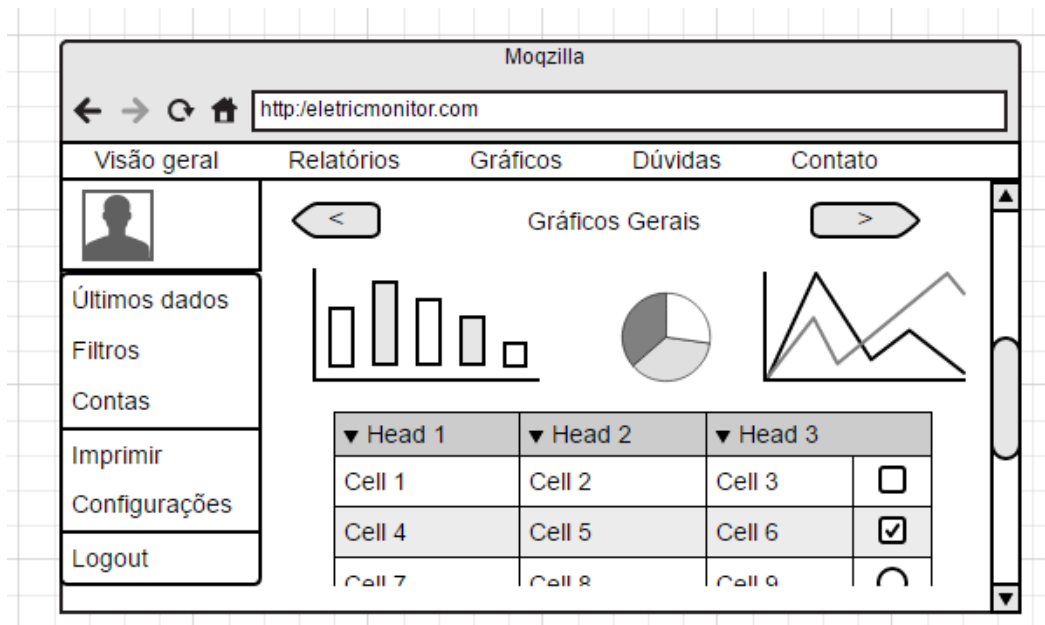
Figura 16 - Wireframe tela de login



Fonte – Elaborado pelos autores

Abaixo, na **Figura 17**, segue *wireframe* do painel principal do usuário

Figura 17 - Wireframe painel principal da aplicação



Fonte – Elaborado pelos autores

2.5.7 Modelagem do banco de dados

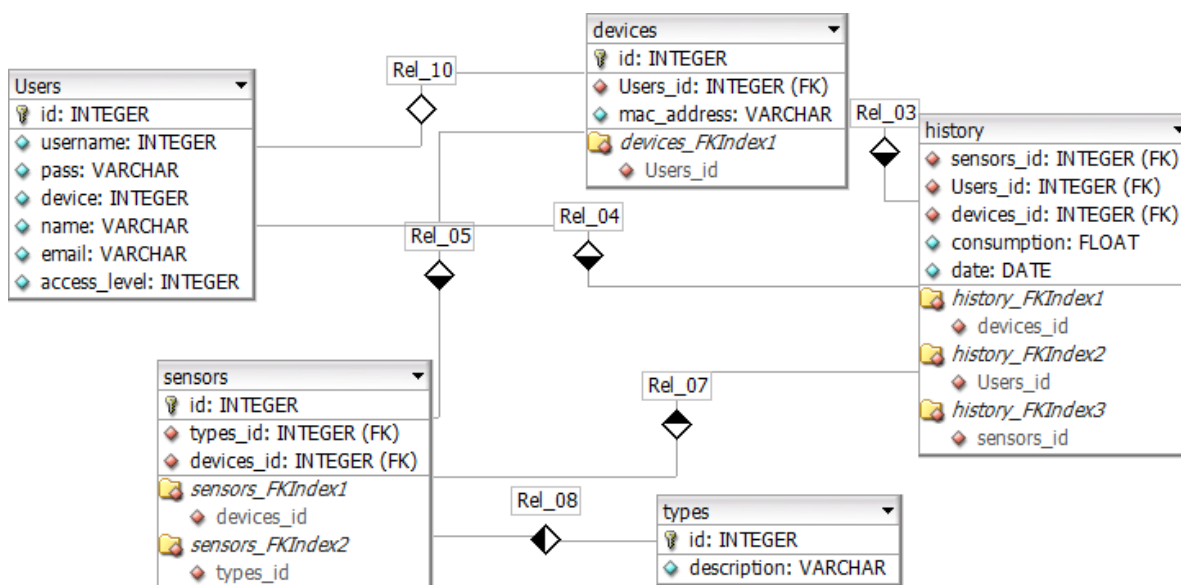
Qualquer aplicação que precise de armazenamento de dados para futuras consultas necessita de uma forma de armazenamento de dados, e quando falamos de aplicação *web* precisamos de um banco de dados para realizar esta persistência dos dados.

Neste cenário utilizamos o MySQL para esta tarefa e precisamos antes de iniciar qualquer desenvolvimento de código, criar um modelo de banco de dados.

O banco de dados da aplicação precisou ser esboçado primeiramente na mão e depois modelado utilizando uma ferramenta adequada para este recurso. Optou-se por utilizar o DBDseigner 4 disponível para download gratuitamente em <http://fabforce.net/downloadfile.php>.

Segue, na **Figura 18**, o modelo utilizado no desenvolvimento da aplicação.

Figura 18 - Modelamento do banco de dados



Fonte – Elaborado pelos autores

Este modelo foi seguido e implementado, porém algumas futuras mudanças serão necessárias quando novos recursos e funcionalidades forem elaboradas e liberadas na solução final.

Para fase de desenvolvimento a base de dados foi mantida em computadores locais e ao final exportada para uma instancia *online*.

2.5.8 Desenvolvimento da API RESTful

Com o ambiente local instalado, o ambiente da Raspberry, os *wireframes* e modelo de dados foi iniciado o desenvolvimento da API RESTful que realiza a integração entre *hardware*, aplicação *web* e banco de dados.

Inicialmente foi feita a instalação e configuração do *framework* ExpressJS através da linha de comando exibida no **Quadro 14**.

Quadro 14 - Instalação ExpressJS

```
pi@raspberrypi: ~ $ npm install -g express
```

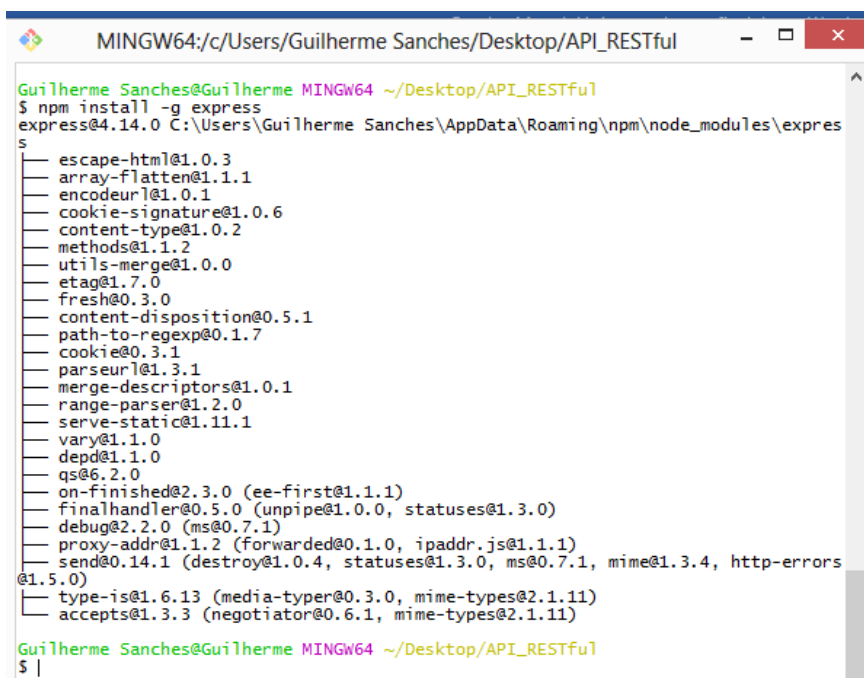
Fonte – Elaborado pelos autores

O ExpressJS foi instalado como um módulo global através do parâmetro “-g”, conforme mostrado na **Figura 19** - Resultado instalação ExpressJS.

O Express fornece uma camada leve de recursos fundamentais para aplicativos da *web*, sem atrapalhar os recursos do Nodejs já existentes.

Através deste *framework* se pôde escrever códigos modularizados, de forma rápida, flexível, utilizando *middlewares* fornecidos pela ferramenta. Sua forma de trabalhar com rotas REST foi de principal importância para gerar resultados satisfatórios ao final do desenvolvimento. E sua documentação repleta de recursos serviu de consulta diversas vezes durante a fase de pesquisa, prototipação e desenvolvimento.

Figura 19 - Resultado instalação ExpressJS



```

MINGW64:/c/Users/Guilherme Sanches/Desktop/API_RESTful
Guilherme Sanches@Guilherme MINGW64 ~/Desktop/API_RESTful
$ npm install -g express
express@4.14.0 C:\Users\Guilherme Sanches\AppData\Roaming\npm\node_modules\expres
s
├── escape-html@1.0.3
├── array-flatten@1.1.1
├── encodeurl@1.0.1
├── cookie-signature@1.0.6
├── content-type@1.0.2
├── methods@1.1.2
├── utils-merge@1.0.0
├── etag@1.7.0
├── fresh@0.3.0
├── content-disposition@0.5.1
├── path-to-regexp@0.1.7
├── cookie@0.3.1
├── parseurl@1.3.1
├── merge-descriptors@1.0.1
├── range-parser@1.2.0
├── serve-static@1.11.1
├── vary@1.1.0
├── depd@1.1.0
├── qs@6.2.0
├── on-finished@2.3.0 (ee-first@1.1.1)
├── finalhandler@0.5.0 (unpipe@1.0.0, statuses@1.3.0)
├── debug@2.2.0 (ms@0.7.1)
├── proxy-addr@1.1.2 (forwarded@0.1.0, ipaddr.js@1.1.1)
├── send@0.14.1 (destroy@1.0.4, statuses@1.3.0, ms@0.7.1, mime@1.3.4, http-errors
@1.5.0)
├── type-is@1.6.13 (media-typer@0.3.0, mime-types@2.1.11)
└── accepts@1.3.3 (negotiator@0.6.1, mime-types@2.1.11)
Guilherme Sanches@Guilherme MINGW64 ~/Desktop/API_RESTful
$ |

```

Fonte – Elaborado pelos autores

Depois do *framework* Express, foi necessário iniciar o projeto criando um arquivo de configuração para aplicações NodeJS, o “package.json”. Para criar este arquivo executou-se o comando “npm init” pelo terminal, como exibido no **Quadro 15**.

Quadro 15 - Criação do arquivo package.json

```
pi@raspberrypi: ~ $ npm init
```

Fonte – Elaborado pelos autores

Informou-se os dados que foram pedidos como pode ser visto na **Figura 20**. Após preenchimento dos dados precisou-se digitar a palavra “yes” para confirmar os dados. Este procedimento criou o arquivo “package.json” do projeto com todas as informações referentes a versionamento, palavras chaves, informações sobre licença de conteúdo, informações sobre os autores, bem como dependências e suas respectivas versões.

Figura 20 - Resultado criação arquivo package.json

```

MINGW64:/c/Users/Guilherme Sanches/Desktop/API_RESTful

Guilherme Sanches@Guilherme MINGW64 ~/Desktop/API_RESTful
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (API_RESTful) eletric_monitor
version: (1.0.0) 0.0.1
description: Gerenciador de consumo de energia elétrica residencial
entry point: (index.js) main.js
test command:
keywords: energia consumo arduino raspberrysesanches/eletricmonitor
author: Guilherme Sanches; Jéssica Adrielle do Nascimento
license: (ISC) MIT
About to write to C:\Users\Guilherme Sanches\Desktop\API_RESTful\package.json:
{
  "name": "eletric_monitor",
  "version": "0.0.1",
  "description": "Gerenciador de consumo de energia elétrica residencial",
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/guilhermesanches/eletricmonitor%1B%5BD%1B%5BD%1B%5BD%1B%5BD%1B%5BD%1B%5BF.git"
  },
  "keywords": [
    "energia",
    "consumo",
    "arduino",
    "raspberry"
  ],
  "author": "Guilherme Sanches; Jéssica Adrielle do Nascimento",
  "license": "MIT",
  "bugs": {
    "url": "https://github.com/guilhermesanches/eletricmonitor%1B%5BD%1B%5BD%1B%5BD%1B%5BD%1B%5BD%1B%5BF/issues"
  },
  "homepage": "https://github.com/guilhermesanches/eletricmonitor%1B%5BD%1B%5BD%1B%5BD%1B%5BD%1B%5BD%1B%5BF#readme"
}

Is this ok? (yes) yes

```

Fonte – Elaborado pelos autores

Com o projeto iniciado precisou-se começar o desenvolvimento do primeiro arquivo principal do projeto, o “main.js”.

Este arquivo tem a função de iniciar o servidor da aplicação RESTful. Seu conteúdo pode ser visto no **Código 4**.

Código 4 - Servidor básico ExpressJS

```
1. var express = require('express');
2. var app = express();
3.
4. app.get('/', function (req, res) {
5.   res.send('Hello World!');
6. });
7.
8. app.listen(3000, function () {
9.   console.log('Example app listening on port 3000!');
10. });
```

Fonte – Elaborado pelos autores

Linha 1: Importa o modulo express.

Linha 2: Cria variável app chamando a execução do servidor express.

Linha 4: Criação de uma rota padrão para testar se o servidor foi iniciado corretamente.

Linha 5: Envia uma mensagem para o console quando a rota “/” for chamada pelo *browser*.

Linha 8: Inicia o servidor na porta 3000.

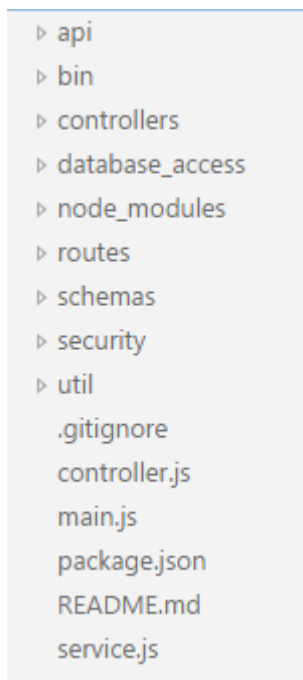
Após todas estas etapas, o ambiente básico para o desenvolvimento da aplicação estava criado. Porém, devido à necessidade da pesquisa, foram adicionados ao sistema os seguintes diretórios:

- **controllers:** Diretório responsável pelos arquivos controladores das *Views*;
- **schemas:** Diretório responsável pelos arquivos de modelo de dados do banco de dados bem como as funções de validação;
- **util:** Diretório contendo funções úteis em várias partes do código, como conversão de datas por exemplo.
- **security:** Diretório criado para configuração de funções e rotinas de validação, autenticação e segurança da API RESTful.
- **routes:** Diretório criado para centralizar todas as rotas de acesso da API RESTful.
- **database_access:** Contém as funções e configurações necessários para acesso ao banco de dados.
- **api:** Diretório que contém as *features* do sistema, bem como sua lógica de negócios.
- **service.js:** Contém serviços genéricos para várias rotas da API.

- **controler.js:** Contém funções de validação genérica para várias rotas da API.

A estrutura de diretórios é demonstrada na **Figura 21**.

Figura 21 - Estrutura de diretórios aplicação web



Fonte – Elaborado pelos autores

Na árvore de diretórios, encontram-se também as pastas *bin* e *node_modules*, as quais contêm respectivamente o arquivo de inicialização do servidor *web* e os módulos NodeJS necessários para a execução do sistema.

O Arquivo README.md contém informações sobre o projeto para ser visualizado na página do GitHub como auxílio aos usuários interessados em conhecer o projeto e suas características.

Todas as dependências utilizadas no projeto estão sendo exibida na **Figura 22**.

Com o decorrer do desenvolvimento, foram surgindo necessidade de novas dependências e as mesmas vieram a serem instaladas e salvas no gerenciador de dependências utilizado NPM.

Figura 22 - Lista de dependências

```

"dependencies": {
  "body-parser": "~1.8.1",
  "cookie-parser": "~1.3.3",
  "cors": "^2.5.3",
  "debug": "~2.0.0",
  "express": "~4.9.0",
  "express-myconnection": "^1.0.4",
  "getmodule": "0.0.3",
  "morgan": "~1.3.0",
  "mysql": "^2.6.1",
  "json-gate": "^0.8.22"
}

```

Fonte – Elaborado pelos autores

2.5.9 Desenvolvimento da aplicação web

A API RESTful possui função de fornecer uma integração entre o hardware e o painel de visualização de dados acessado pelos usuários de forma *online*, portanto foi necessário o desenvolvimento de uma *interface web* para que os usuários das residências possam verificar os dados colhidos pela Arduino, sensor e Raspberry e gerar dados analíticos a partir dos mesmos.

O primeiro passo nessa etapa de projeto foi a criação do diretório raiz e instalação de um projeto inicial usando o *framework* AngularJS na versão estável, 1.5.8.

Para esta instalação necessitou-se previamente da instalação do gerenciador de dependências NPM que é instalado juntamente com o NodeJS, logo, a sessão de preparação do ambiente da API já havia contemplado este requisito.

Dentro de um terminal de comando aberto dentro do diretório raiz, foi digitado o comando exibido no **Quadro 16**.

Quadro 16 - Instalação do AngularJS

```
Gui_sanches@guilherme-PC: ~ $ npm install angular
```

Fonte – Elaborado pelos autores

Este comando criou uma pasta chamada `node_modules` contendo todo o código *core* para que pudéssemos incluí-lo em nossa aplicação. E para este procedimento necessitou-se dentro de uma *tag script* no arquivo `index.html`, incluir o seguinte código.

Código 5 - Tag de inclusão AngularJS

```
1. <script src="/node_modules/angular/angular.js"></script>
```

Fonte – Elaborado pelos autores

Além do AngularJS houve a necessidade de instalação de um módulo específico para controle de rotas em nossa aplicação, este módulo chama-se `angular-route` e para sua instalação executamos o seguinte comando exibido no **Quadro 17**.

Quadro 17 - Instalação angular-route

```
Gui_sanches@guilherme-PC: ~ $ npm install angular-route
```

Fonte – Elaborado pelos autores

Este módulo possui função específica de criar e gerenciar nossas rotas e transições entre elas de forma ágil e prática dentro de uma aplicação baseada em AngularJS.

O **Código 6** exibe algumas rotas utilizadas na aplicação *web* da solução desenvolvida.

Código 6 - Exemplo de rotas utilizando angular-route

```
1. $routeProvider
2.   .when('/', {
3.     templateUrl: 'views/home.html',
4.     controller: 'HomeCtrl',
5.     access: { requiredAuthentication: true }
6.   })
7.
8.   .when('/report', {
9.     templateUrl: 'views/reportDate.html',
10.    controller: 'ReportCtrl',
11.    access: { requiredAuthentication: true }
12.  })
13.
14.   .when('/login', {
15.     templateUrl: 'views/login.html',
16.     controller: 'LoginCtrl',
17.     access: { requiredAuthentication: false }
18.   })
19.
20.   .when('/signup', {
21.     templateUrl: 'views/signup.html',
```

```
22.         controller: 'SignupCtrl',  
23.         access: { requiredAuthentication: false }  
24.     })  
25.  
26.     .otherwise({  
27.         redirectTo: '/'  
28.     });
```

Fonte – Elaborado pelos autores

Outro ponto que merece destaque é a instalação da biblioteca jQuery para prover uma agilidade em lidar com recursos do HTML, e ela foi instalada pelo comando listado no **Quadro 18**.

Quadro 18 - Instalação jQuery

```
Gui_sanches@guilherme-PC: ~ $ npm install jquery
```

Fonte – Elaborado pelos autores

E para gerar uma *build* e testes durante desenvolvimento, necessitou-se a instalação de um servidor HTTP local para executar o projeto. Dentro os diversos existentes, deu-se preferência ao servidor http-server, e o mesmo foi instalado pelo comando como segue abaixo no **Quadro 19**.

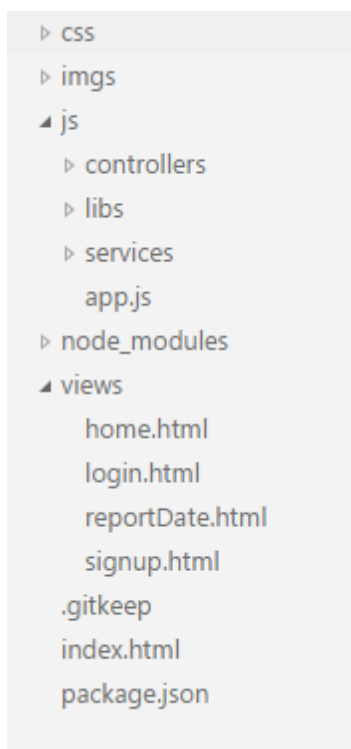
Quadro 19 - Instalação servidor HTTP

```
Gui_sanches@guilherme-PC: ~ $ npm install -g http-server
```

Fonte – Elaborado pelos autores

Como pode ser visto, o módulo foi instalado de forma global, para que o mesmo não seja alocado dentro de nossa pasta node_modules e possa ser usado em qualquer outro projeto que necessite de um servidor http.

Ao final a estrutura do projeto ficou da seguinte forma, como exibido na **Figura 23**.

Figura 23 - Estrutura de pastas da aplicação

Fonte – Elaborado pelos autores

Após estruturação de pastas, módulos e configuração o desenvolvimento precisou adotar uma prática de layout responsivo seguindo os padrões do Google Material Designer, este padrão é adotado para que haja uma melhor usabilidade do usuário final com o sistema e o mesmo possa ser manutenível de forma padrão e simples, por utilizar conceito de classes de css.

Este recurso foi disponível através da instalação do angular-material que é um modulo que nos traz todos componentes do material designer já para AngularJS.

O comando listado no **Quadro 20** fez a instalação do angular-material no projeto:

Quadro 20 - Instalação Angular Material

```
Gui_sanches@guilherme-PC: ~ $ npm install angular-material
```

Fonte – Elaborado pelos autores

Para utilizar os componentes, foi preciso acessar o site do angular-material, disponível em: <https://material.angularjs.org/>. Selecionar quais componentes eram necessários pelo menu lateral esquerdo, como mostra a **Figura 24**.

Figura 24 - Menu de componentes Angular Material



Fonte – Elaborado pelos autores

Após selecionar o componente necessário, o código do mesmo aparece na tela para ser copiado e incluído ao projeto. Estes recursos trouxeram ganho de tempo durante desenvolvimento por já serem componentes adaptáveis automaticamente à vários tamanhos de telas e também por serem estilizados de forma padrão usando conceitos de Material Designer.

3 DISCUSSÃO DE RESULTADOS

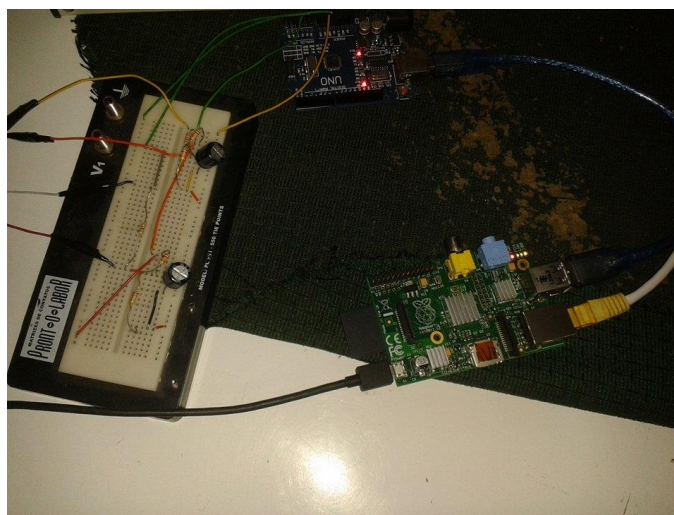
Neste capítulo são apresentados e discutidos os resultados obtidos pela pesquisa por meio do desenvolvimento das tecnologias empregadas no trabalho, com o intuito de demonstrar os pontos fortes e fracos encontrados.

Este trabalho teve como objetivo desenvolver uma aplicação *web* capaz de gerenciar o consumo de energia elétrica residencial. Esta aplicação tinha por objetivo coletar a energia elétrica dos setores¹⁴ de uma residência através de placas eletrônicas com a utilização de sensores; armazenar os dados em um banco de dados externo e demonstrar as informações mensuradas para os usuários em um ambiente *online*.

Para alcançar os objetivos mencionados foram indispensáveis diversas pesquisas para modelagem da arquitetura utilizada e escolha das ferramentas e tecnologias que atenderiam da melhor forma as necessidades do trabalho.

A arquitetura desenvolvida contou com total desacoplamento entre as camadas refletindo resultados positivos em relação à manutenibilidade do sistema. Tendo sido definida a arquitetura e as tecnologias utilizadas foram feitas as ligações dos equipamentos e o protótipo do produto final, como pode ser visto na **Figura 25**.

Figura 25 - Conexão entre as placas



Fonte – Elaborada pelos autores

¹⁴ Definimos setores cada conjunto da residência que está conectado a um mesmo disjuntor elétrico. Alguns casos os setores representam: lâmpadas, tomadas e chuveiros; outros casos representam os cômodos da residência, por exemplo: sala, cozinha, quartos e banheiros.

Com as conexões feitas o próximo passo foi desenvolver o código que seria gravado na plataforma de desenvolvimento Arduino. Este código teria por objetivo escrever na saída serial da Arduino o valor da corrente que circulava pela extensão que utilizamos para teste. Ao término, os resultados tiveram variações se comparado ao valor especificado pelo fabricante de cada equipamento que havia sido ligado.

Com estes resultados insatisfatórios, pesquisou-se referências em outros trabalhos nesta área, o qual nos trouxe alternativas para tais problemas. Como havíamos nos apoiado em referências de outros trabalhos semelhantes que realizavam as mesmas medições, não notamos que os sensores utilizados em nossa pesquisa possuíam um circuito elétrico diferente dos quais haviam sido implementados pelos outros trabalhos. Logo, foi necessário alterar o circuito elétrico para que o mesmo fosse capaz de atender o equipamento utilizado. Então os dados lidos começaram a apresentar resultados mais satisfatórios se comparado aos dos fabricantes, porém, ainda constavam variações maiores que 10% em alguns casos.

Este problema foi solucionado efetuando a devida calibração que o sensor de 100A necessitava como pode ser visto no Capítulo 2, seção 1.4.4 deste trabalho. Após os procedimentos terem sido executados, os resultados obtidos passaram a ter variações menores que 3%, ou seja, ficaram dentro da variação máxima permitida pelo fabricante do sensor de corrente.

Observe na **Figura 26** que a média dos resultados obtidos apresentaram uma porcentagem menor que a porcentagem que o fabricante do sensor assegura.

Figura 26 - Medições comparativas dos equipamentos de teste

Dispositivo	Corrente A	Corrente B	Corrente C	Variação
Secador de cabelo	10	8,90	8,80	1,124%
Ferro de passar	10	9,35	9,23	1,283%
Ventilador	0,5	0,47	0,45	4,255%
Panela elétrica	3,15	3,14	3,1	1,274%
Chuveiro	15A	15,1	14,72	2,517%

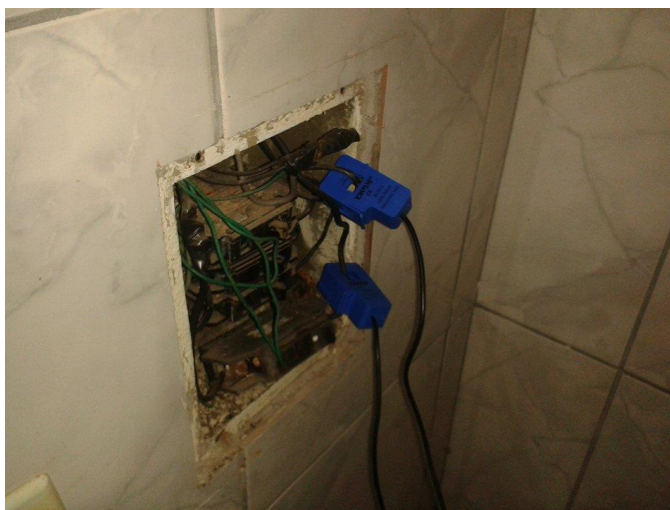
Corrente A	Fabricante	Media Variação
Corrente B	Alicate Amperímetro	2,0906%
Corrente C	Sensor SCT 013-000	Variação Fabricante
		3%

Fonte: Desenvolvida pelos autores

Outro ponto que pode ser analisado com os resultados da Figura 25 é que a menor corrente, lida do equipamento ventilador, apresenta uma taxa de variação maior. Este resultado se apoia na teoria de que como existe uma corrente de ruído que fica em torno de 0.1 a 0.4 ampères, as correntes menores que 1 ampère sofre maiores variações, pois são somadas a estes ruídos. Logo, correntes maiores não representam dados significativos de variações.

Como os sensores são conectados aos disjuntores elétricos, os mesmos tendem a sempre medirem correntes maiores que 1 ampère, por este fato, foram obtidos resultados satisfatórios nas leituras. A **Figura 27** representa a forma como os sensores são ligados nos disjuntores.

Figura 27 - Conexão entre os sensores e os disjuntores



Fonte – Elaborada pelos autores

A ligação dos sensores ocorre de forma não invasiva, ou seja, não é necessária qualquer inclusão do sensor com a rede elétrica, o resultado representou uma forma mais prática e segura de se manusear com tensões elétricas.

Após as leituras estarem dentro das taxas de variações aceitáveis foi obtida a persistência dos dados de modo *online*, através da conexão da *Raspberry* com nossa *API RESTful* desenvolvida, responsável por receber os dados na forma de *POST* e salvar as informações no banco de dados *MySQL* a cada 30 minutos.

Figura 28 - Dados persistidos no banco online

sensors_id	Users_id	devices_id	consumption	date
0	1	123456	0.357481	2016-09-14 17:51:49
1	1	123456	1.08963	2016-09-14 17:51:49
0	1	123456	0.357771	2016-09-14 18:21:45
1	1	123456	1.08992	2016-09-14 18:21:45
0	1	123456	0.357981	2016-09-14 18:51:50
1	1	123456	1.09036	2016-09-14 18:51:50
0	1	123456	0.358359	2016-09-14 19:21:50
1	1	123456	1.09068	2016-09-14 19:21:50

Fonte – Elaborada pelos autores

A **Figura 28** demonstra os dados lidos entre às 17h51 e 19h:21 do dia 14 de setembro de 2016. Pode-se observar que existem 2 sensores, sendo eles: 0 e 1, conectados ao usuário com identificador 1. Uma placa *Raspberry* com identificador 12356 e seu respectivo valor de consumo que já é salvo na unidade quilowatts/hora.

Outra conexão entre a *Raspberry* e a API é realizada a cada 20 segundos, atualizando o valor total até este momento em quilowatts/hora. Como pode ser visto na **Figura 29** e **Figura 30**.

Figura 29 - Consumo em 16 de setembro de 2016 às 12h e 09 minutos

user_id	last_consumption	date
1	7.104446030500022	2016-09-16 12:09:52

Fonte – Elaborada pelos autores

Figura 30 - Consumo em 16 de setembro de 2016 às 12h e 10 minutos

user_id	last_consumption	date
1	7.104441032000023	2016-09-16 12:10:13

Fonte – Elaborada pelos autores

A **Figura 29** apresenta o valor total consumido em quilowatts/hora pelos 2 sensores até as 12 horas 09 minutos e 52 segundos do dia 16 de setembro e a **Figura 30** também, só que 20 segundos após a última medição. Nestas figuras também pode-se verificar que houve uma variação de tempo de 21 segundos, sendo que foi programado apenas 20 segundos, esta

condição deve-se ao fato de que o tempo da requisição pode variar de acordo com a internet local, e isto, pode representar uma variação na data salva no banco de dados. Porém, este resultado não interfere no objetivo final da pesquisa.

Caso exista alguma falha na leitura do *hardware* uma função de diagnóstico demonstrou ser eficaz ao exibir o *status* de cada sensor, bem como sua data de última atualização. Isto trouxe à solução um recurso de grande importância e a **Figura 31** exibe o *layout* deste módulo de diagnóstico.

Figura 31 - Ferramenta de diagnóstico

Diagnóstico			
Sensor	Setor Residencial	Última atualização	Status
1	Lâmpadas e tomadas	25/9/2016, 18:00:38	✓
2	Chuveiro e cozinha	25/9/2016, 18:00:38	✓

Fonte – Elaborada pelos autores

Quando um sensor por alguma razão parou de medir, como falta de acesso à internet, o status fica com um ícone “X” demonstrando ao usuário que existe algum problema na solução e que um contato com suporte deve ser realizado.

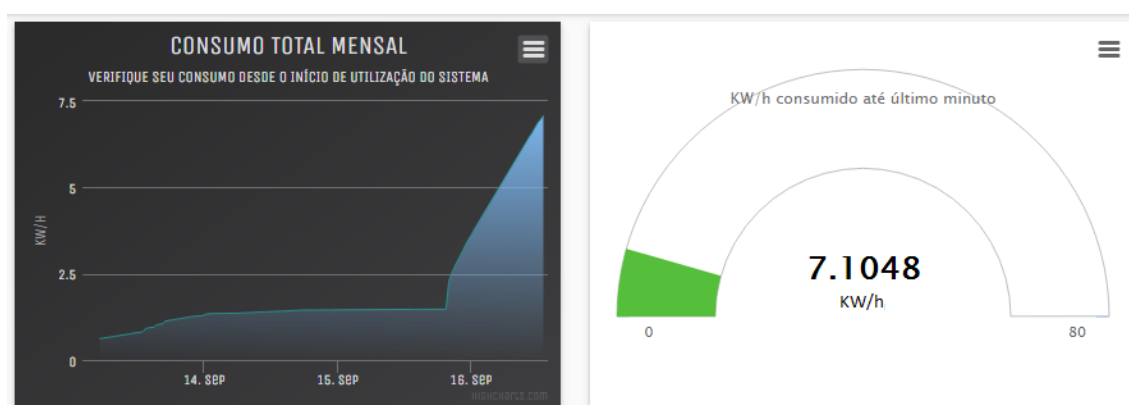
O último resultado alcançado, refere-se à apresentação dos dados lidos e persistidos de forma *online* ao usuário final através de uma *interface web* a qual o usuário possui um acesso que o permite realizar *login* de forma segura. Como pode ser visto na **Figura 32**.

Figura 32 - Painel de *login*

Fonte – Elaborada pelos autores

A forma de login segue padrões de segurança através de *token authentication* que expira a cada dia. Esta validação da API assegura que usuários mal-intencionados tenham maiores dificuldades caso queiram acessar dados dos clientes desta solução proposta.

Após realizarem *login* são apresentados os resultados através de gráficos que representam valores em quilowatts/hora desde o início da implantação do sistema.

Figura 33 - Gráficos de apresentação de resultados

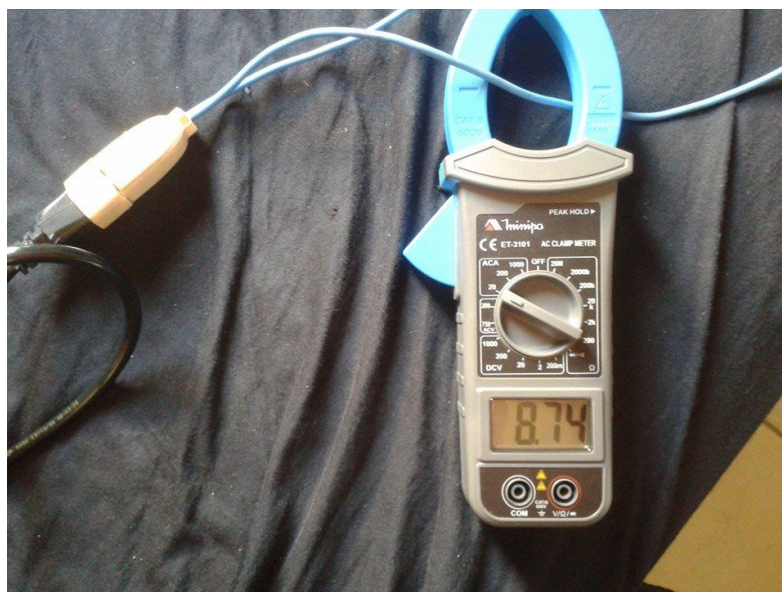
Fonte – Elaborada pelos autores

A **Figura 33** mostra dois gráficos representando os valores lidos. O primeiro mostra os dados lidos desde o início do monitoramento e o segundo os valores até os últimos 20 segundos.

Pode-se notar que no primeiro gráfico (Figura 32) houve um pico ao final do dia 16 de setembro de 2016, este pico ocorreu, pois ainda estavam sendo realizados os testes no alicate amperímetro.

Na **Figura 34**, pode-se observar o alicate amperímetro fazendo a mensuração dos equipamentos ligados na extensão.

Figura 34 - Testes e validações com alicate amperímetro



Fonte – Elaborada pelos autores

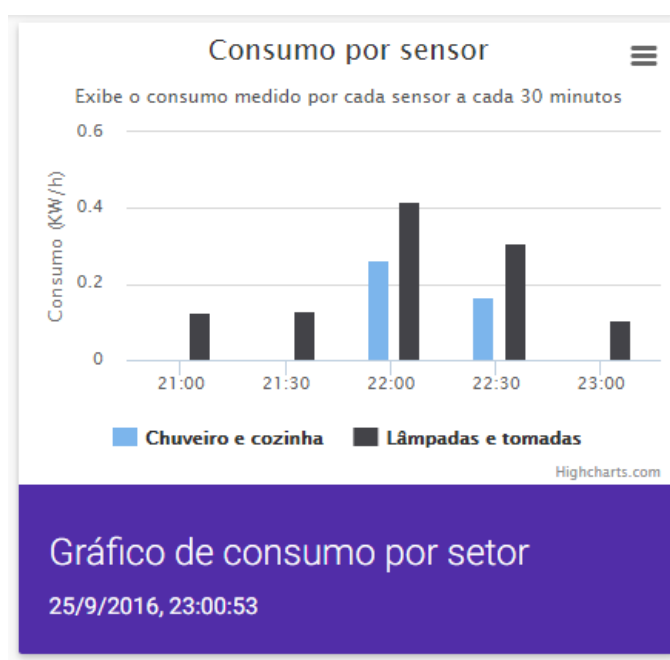
Após este período de testes e validações, os sensores foram conectados aos disjuntores da residência e então os valores medidos se elevaram, devido a mensuração estar sendo feita por toda parte elétrica da residência.

Na **Figura 35** exibe-se os resultados lidos de cada sensor a cada intervalo de 30 minutos. Ao passar o mouse em cima de cada barra exibida no gráfico o mesmo exibe uma caixa de texto informativa contendo os valores de consumo desde a última mensuração até o momento desta outra mensuração de cada sensor. Abaixo exibe-se uma legenda informando ao usuário a cor de cada sensor representando cada um, um setor residencial.

Figura 35 - Gráfico de consumo por sensor

Fonte – Elaborada pelos autores

Enquanto que na **Figura 36**, mostra-se os dados lidos das 21 às 23 horas do mesmo dia.

Figura 36 - Gráfico de consumo por sensor

Fonte – Elaborada pelos autores

Ao analisar os gráficos, pode-se notar que entre as 21h30 e 22h30 houve um aumento grande no sensor do chuveiro, conclui-se, portanto, que neste período um ou mais usuários da residência utilizaram o chuveiro elétrico. Através destes gráficos detalhados, pode-se criar rotinas de comparação entre cada pessoa da residência, criar períodos em que são gastos maior valor energético e diversos outros recursos.

Por fim, demonstrou-se que o trabalho realizado atingiu seus objetivos, pois foi possível monitorar o custo e consumo de energia elétrica de forma prática e instantânea, fazendo o melhor uso possível dos recursos e componentes disponíveis, atingindo resultados bastante satisfatório para os objetivos deste trabalho. No que se refere às divergências de medição, é aceitável que o dispositivo não atinja sua otimização ideal, pois se trata de um primeiro protótipo.

A central de monitoramento desenvolvida foi a materialização de uma ideia que pode vir a se tornar um produto inovador e prático para as residências.

REFERÊNCIAS

AGUILAR, L. J. **Fundamentos de Programação - Algoritmos, estruturas de dados e objetos**. 3ª Edição. ed. São Paulo: AMGH Editora, 2008.

ANGULARJS. O que é AngularJS. **ANGULARJS**, 2016. Disponível em: <<https://docs.angularjs.org/guide/introduction>>. Acesso em: 11 Agosto 2016.

ARDUINO Introduction. **Arduino**, 2016. Disponível em: <<https://www.arduino.cc/en/Guide/Introduction>>. Acesso em: 10 Março 2016.

ASHTON, K. That "Internet of Things" Thing. **RFID Journal**, 2009. Disponível em: <<http://www.rfidjournal.com/articles/view?4986>>. Acesso em: 20 Março 2016.

CAELUM. Rest - Web Ágil com VRaptor, Hibernate e Ajax. **CAELUM**, 2016. Disponível em: <<https://www.caelum.com.br/apostila-vraptor-hibernate/rest/#11-3-o-triangulo-do-rest>>. Acesso em: 02 Agosto 2016.

DAMAS, L. **Linguagem C**. 10. ed. Rio de Janeiro: LTC, 2007.

EVANS, D. **The internet of things**: how the next evolution of the internet is changing everything. [S.l.]: CISCO white paper, 2011. 11 p.

EVANS, M.; NOBLE, J.; HOCHENBAUM, J. **Arduino em ação**. São Paulo: Novatec Editora, 2013.

FREMAN, A. **Pro AngularJS**. Nova Iorque: Apress Media LLC, 2014.

GREEN, B.; SESHADRI, S. **AngularJS**. Sebastopol: O'Reilly Media, 2013.

HIGHCHARTS. Highcharts Documentation. **Highcharts**, 2016. Disponível em: <<http://www.highcharts.com/docs>>. Acesso em: 26 Julho 2016.

KENDE, M. Global Internet report. **Geneva**: Internet Society, 2014. Disponível em: <<http://www.internetsociety.org/doc/global-internet-report>>. Acesso em: 21 Março 2016.

KLOTZ, D. C for Embedded Systems Programming. **NXP**, 11 Novembro 2010. Disponível em: <http://www.nxp.com/files/training/doc/dwf/AMF_ENT_T0001.pdf>. Acesso em: 10 Março 2016.

LACERDA, F.; LIMA-MARQUES, M. Da necessidade de princípios de Arquitetura da Informação para a Internet das Coisas. **Perspectivas em Ciência da Informação**, Belo Horizonte, 20, Junho 2015. 158-171.

MATERIALIZE. Documentação - Materialize. **Materialcss**, 2016. Disponível em: <<http://materializecss.com/>>. Acesso em: 21 Março 2016.

MONGODB. MongoDB for GIANTI Ideas. **MongoDB**, 2016. Disponível em: <<https://www.mongodb.com/>>. Acesso em: 01 Agosto 2016.

MOREIRA, R. H. O que é Nodejs? **NodeBr**, 2013. Disponível em: <<http://nodebr.com/o-que-e-node-js/>>. Acesso em: 20 Julho 2016.

MOZILLA DEVELOPER NETWORK. **JSON**, 2014. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/JSON>>. Acesso em: 20 Março 2016.

MOZILLA DEVELOPER NETWORK. **JavaScript**, 2016. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>>. Acesso em: 22 Abril 2016.

ORACLE. MySQL | O Banco de Dados de Código Aberto Mais Popular. **ORACLE**, 2016. Disponível em: <<http://www.oracle.com/br/products/mysql/overview/index.html>>. Acesso em: 20 Fevereiro 2016.

PEREIRA, C. R. **Node.js**: Aplicações web real-time com Node.js. São Paulo: Casa do Código, 2015.

RASPBERRY PI, 2014. Disponível em: <<https://www.raspberrypi.org/help/what-is-a-raspberry-pi/>>. Acesso em: 20 Março 2016.

SANTOS, W. R. D. **Univale**, 2009. Disponível em: <<http://www.univale.com.br/unisite/mundo-j/artigos/35RESTful.pdf>>. Acesso em: 23 Fevereiro 2016.

SCHILDT, H. **Linguagem C**: Guia do Usuário. São Paulo: MC Graw Hill, 1986.

SILVA, M. S. **JavaScript**: Guia do Programador. São Paulo: Novatec Editora, 2010.

SILVA, M. S. **JQuery**: A Biblioteca do Programador JavaScript. 3. ed. São Paulo: Novatec Editora, 2014.

SILVEIRA, G. Internet das Coisas - O que é IoT? **Bluelux**, 2016. Disponível em: <<http://www.bluelux.com.br/internet-das-coisas-iot/>>. Acesso em: 31 Julho 2016.

SOARES, J. O que é MongoDB e porque usá-lo? **Código Simples**, 2016. Disponível em: <<http://codigosimples.net/2016/03/01/o-que-e-mongodb-e-porque-usa-lo/>>. Acesso em: 10 Agosto 2016.

SOUZA, F. Arduino UNO - Conheça os detalhes de seu hardware. **Embarcados**, 2016. Disponível em: <<http://www.embarcados.com.br/arduino-uno/>>. Acesso em: 20 Fevereiro 2016.

STAMFORD, C. Gartner Says the Internet of Things installed base will grow to 26 billion units by 2020. **Gartner**, 2013. Disponível em: <<http://www.gartner.com/newsroom/id/2636073>>. Acesso em: 21 Março 2016.

UPTON, E.; , G. H. **Raspberry pi**: Manual do usuário. Tradução de Celso Roberto Paschoa. São Paulo: Novatec Editora, 2013.

APÊNDICE